Strongly Secure Authenticated Key Exchange in the Standard Model

Zheng Yang

Dissertation zur Erlangung des Grades eines Doktor-Ingenieurs
der Fakultät für Elektrotechnik und Informationstechnik
an der Ruhr-Universität Bochum

Bochum, 2013
Abstract

Nowadays many crucial network applications rely on the existence of a confidential channel established by authenticated key exchange (AKE) protocols over public networks. With the rapid development of cyber technology, novel attacks to cryptosystem emerge in an endless stream. This has also led to the development of AKE solutions to provide increasingly stronger security guarantees. In this thesis we focus on provision of practical constructions for AKE protocols which are provably secure in a strong sense without resorting to random oracles.

We first we present three new efficient compilers to generically turn passively secure key exchange protocols (KE) into authenticated key exchange protocols (AKE) where security also holds in the presence of active adversaries. Our compilers are not only a useful tool for the design of new AKE systems with many additional security properties in a modular and less error-prone fashion, but they also help to relax the assumptions on existing, practical key exchange mechanisms which are not known to be provably secure AKE protocols. Security of our compilers is shown in a strong modified CK model where the adversary is allowed to reveal either long-term secret key or state information of the protocol participants.

On the second, we study the open problem on constructing eCK secure two party AKE protocol without random oracles and NAXOS trick. A generic construction satisfying those requirements is given based on well-known cryptographic primitives following the guideline of efficiency. Then a concrete protocol is proposed which is the first eCK secure protocol in the standard model under both standard assumptions and post-specified peer setting (i.e. without knowing any cryptographic information about its communication peer). Both proposed schemes can be more efficiently implemented with secure device than previous works which are eCK secure in the standard model, where the secure device might be normally used to store the long-term private key and to implement codes of protocol which need to be resilience of state leakage.

Finally, we generalize our one-round two party AKE construction to group case that yields an efficient tripartite AKE protocol based on bilinear maps and a candidate group AKE protocol (with more than three members) based on multilinear maps. Up to now they are the first solutions which can be proved secure in the g-eCK model without random oracles. Meanwhile we make first step to show how to simplify the security proof for one-round group AKE under g-eCK model.
I would like you to express my deep and sincere gratitude to my supervisor Prof. Joerg Schwenk who gives me the great opportunity to pursue my research dream at Horst Goertz Institute for IT-Security. Without his guidance, encouragement and support, I could never have accomplished this serious undertaking. I am grateful as well to my thesis reviewer Prof. Mark Manulis for giving me invaluable advice and comments. Thanks also go to my colleagues of Chair of Network- and Data Security. I have benefited a lot from interesting conversations and correspondence with them. I would like to especially thank Sven Schaege for numerous fruitful discussions and intensive joint work on modular compilers for authenticated key exchange. Further, I would like to thank Yong Li for supporting me, as well as for quite a few joint works on security models and cryptographic protocols. I am also grateful to Tibor Jager for everything I have learned from him. Additionally, I would like to thank Florian Kohlar and Florian Feldmann from whom I got a lot of valuable suggestions, ideas and generous assistance. Finally, I would like to express my gratitude to my parents PeiXin Yang and Yuanju Wei, my wife Li Huang and our lovely daughter Qiaoyi Yang for bringing eternal sunshine to my life. Their unconditional support, encouragement, quiet patience and unwavering love were undeniably the bedrock of this dissertation.
List of Figures

2.1 General Two-move KE Protocol ........................................ 14
4.1 Attack on JKSS AKE Compiler due to Leakage of States .......... 48
4.2 AKE Compiler from Signature ........................................... 50
4.3 AKE Compiler from PKE and One-time MAC .......................... 56
4.4 AKE Compiler from KEM and MAC .................................... 62
5.1 General OR2AKE Protocol ............................................... 71
5.2 Generic One-round AKE Protocol from KE, KEM and NIKE ....... 80
5.3 Pairing-based AKE Protocol under Standard Assumptions ......... 87
6.1 Reductions of g-eCK-Freshness for One-round Tripartite AKE .... 103
6.2 One-round Tripartite AKE Protocol ................................... 106
## Contents

1 Introduction ................................. 1
   1.1 Towards Strong Security Requirements for AKE ............... 2
   1.2 On Security Models for AKE .......................... 4
      1.2.1 On Models for Two Party AKE .................... 5
      1.2.2 On Models for Group AKE ....................... 7
      1.2.3 Selecting Appropriate Models for AKE Protocols .......... 8
      1.2.4 Connection Between Practical Implementation and Theoretical Security ........................................... 9
      1.2.5 Constructions in the Standard Model ................. 10
   1.3 Structure of This Thesis .......................... 11

2 Preliminaries ............................... 12
   2.1 Negligible Functions ............................. 12
   2.2 Key Exchange Protocols .......................... 13
      2.2.1 General KE Protocols ............................ 17
   2.3 Digital Signature Schemes .......................... 18
   2.4 Public Key Encryption Schemes ...................... 19
   2.5 Key Encapsulation Mechanism ........................ 20
   2.6 Non-Interactive Key Exchange Protocols ................ 21
   2.7 Message Authentication Code ........................ 23
   2.8 Collision-Resistant Hash Functions .................... 24
   2.9 Target Collision-Resistant Hash Functions .............. 24
   2.10 Pseudo-Random Functions ........................... 25
   2.11 Min-entropy and Strong Randomness Extractors ............ 25
   2.12 Tag-based Authentication Schemes ..................... 26
   2.13 Weak Programmable Hash Functions .................... 28
   2.14 Bilinear Groups .................................. 29
   2.15 Multilinear Groups ................................ 29
   2.16 Complexity Assumptions .......................... 30
      2.16.1 Bilinear Decisional Diffie-Hellman Assumption .. 30
      2.16.2 Cube Bilinear Decisional Diffie-Hellman Assumption 30
      2.16.3 n-Multilinear Decisional Diffie-Hellman Assumption 31
## 3 Security Models for Authenticated Key Exchange

3.1 Execution Environment ............................................ 32
3.2 Adversarial Model .................................................. 33
3.3 Secure Two Party AKE Protocols ................................. 36
3.4 Secure Group AKE Protocols ...................................... 38
3.5 Remarks on Models .................................................. 38

## 4 New Modular Compilers for Authenticated Key Exchange

4.1 Background .......................................................... 41
   4.1.1 Motivations .................................................... 41
   4.1.2 Contributions ................................................ 42
   4.1.3 Related Works for AKE Compilers ......................... 45
   4.1.4 The Attack due to Leakage of Session States from KE Protocol 47
4.2 Compiler for AKE Protocol from Signature ...................... 49
   4.2.1 Protocol Description ....................................... 49
   4.2.2 Security Analysis ........................................... 51
4.3 Compiler for AKE Protocol From PKE and (One-time) MAC ..... 55
   4.3.1 Protocol Description ....................................... 55
   4.3.2 Security Analysis ........................................... 57
4.4 Compiler for AKE Protocol From KEM and MAC ............... 61
   4.4.1 Protocol Description ....................................... 61
   4.4.2 Security Analysis ........................................... 63
4.5 Take Care of the Session Identifier .............................. 66

## 5 One-round Two Party Authenticated Key Exchange

5.1 Background .......................................................... 70
   5.1.1 Notations for One-round Two Party AKE ................... 70
   5.1.2 Motivations .................................................... 72
   5.1.3 Contributions ................................................ 73
   5.1.4 Related Works ................................................ 74
   5.1.5 Simplify the Security Proof for One-round Two Party AKE in the eCK Model ........................................ 76
5.2 A Generic One-round Two Party AKE Protocol from KE, KEM and NIKE 77
   5.2.1 Protocol Description ....................................... 80
   5.2.2 Security Analysis ........................................... 82
5.3 A One-round Two Party AKE Protocol under Standard Assumptions 86
   5.3.1 Protocol Description ....................................... 87
   5.3.2 Performance Improvement ................................... 89
   5.3.3 Security Analysis ........................................... 90
5.4 Comparisons .......................................................... 94
6 One-round Group Authenticated Key Exchange 96
  6.1 Background ........................................ 96
    6.1.1 Notions for One-round GAKE ................. 96
    6.1.2 Motivations .................................... 97
    6.1.3 Contributions .................................. 98
    6.1.4 Related Works .................................. 99
    6.1.5 Simplify the Security Proof for One-round GAKE in the g-eCK model .......... 100
  6.2 A Tripartite AKE Protocol from Bilinear Maps ............. 104
    6.2.1 Protocol Description ......................... 105
    6.2.2 Security Analysis ............................. 108
  6.3 A GAKE Construction from Multilinear Maps ............. 114
    6.3.1 Protocol Description ......................... 114
    6.3.2 Security Analysis ............................. 116

7 Conclusion and Future Work 121

Bibliography 122

Curriculum Vitae 135
1 Introduction

Authenticated Key Exchange (AKE) is a fundamental cryptographic primitive which forms a crucial component in many network protocols. The AKE problem is stated that two or more parties wish to share keying information in secret over an open network with an assurance that no party other than these intended communication parties can possibly compute the common shared secret key. Meanwhile the situation where three or more parties share a secret key is often called group (conference) keying. With respect to the authentication issue in AKE protocols, it typically requires a set-up phase whereby authentic and possibly secret initial keying material is distributed. For example in a public key infrastructure (PKI) based AKE protocol, each party possesses long-term private key associated with corresponding public key, where the public key is expected to be certified with a party’s identity (ID) at certificate authority (CA) that is typically described as a trusted third party.\(^1\) The agreed session key is usually authenticated relying on the long-term key of each participant, which is used to secure underlying application data. Different session keys might be generated in order to satisfy the requirement of different security mechanisms within the underlying application. For an application requiring both confidentiality and integrity, two different session keys might be derived for symmetric encryption scheme and message authentication code scheme.

The root of asymmetric key exchange dates back to 1976 when Diffie and Hellman published their most stunning invention on public key cryptography. The original Diffie-Hellman protocol does not provide protection against impersonation attacks due to lack of authentication. With the rapid development of cyber technology, novel attacks to cryptosystem emerge in an endless stream. Early design of the AKE protocols (see \[BWM99\] and \[Man06\] for surveys) may fail to provide security guarantees to recently state-of-art attacks aiming to secure devices or softwares such as malwares and side channel attacks \[KJJ99, CCD00\]. In a nutshell, the goal of this thesis is to make strides toward the provision of practical solutions for AKE problems which are provably secure in a strong sense without resorting to random oracles \[BR93\].

\(^1\)On receiving a key registration request (ID, PK), the CA generates a certificate \textit{cert} which consists of a signature on (ID, PK) using its private key.
1 Introduction

1.1 Towards Strong Security Requirements for AKE

In practical we might expect to obtain security assurances from AKE protocols concerning access control, secure communication and processing and so on. Nevertheless most of them may fall into the following two common classes of the security goals of AKE in a distrusted system: entity authentication and key distribution.

**Entity Authentication & Key Distribution.** The notion of entity authentication for a two party authentication protocol is normally defined as a process by which a party Alice in the distributed environment gains confidence in the identity of a communication partner Bob, namely Alice is assured of the identity of Bob through the obtained message and that Bob was actively involved in the generation of this message. In the context of AKE protocol, the entity authentication process is typically associated with the distribution of a ‘session key’ which can be later used by its owners for message confidentiality, integrity and other security purposes. In a key distribution process, a shared secret is derived by two or more parties as a function of information contributed by or associated with each of these parties which satisfy properties of key authentication and confidentiality. The key authentication property [MvOV96, Def. 12.6] addresses that each legitimate protocol participant is assured that no other party except for possibly unidentified session participants may learn the established session key; and the confidentiality [MvOV96, Chapter 1] is a requirement on “keeping information secret from all but those who are authorized to see it”. It is straightforward to see that key authentication property can be realized by entity authentication process but not limited to it. A related notion is called *implicit key authentication* [MvOV96, Chapter 12] only requires that one party is assured the session key is shared only with its intended communication partners without assurance that the actions of key distribution are involved whatsoever by its partners. Thus the authentication issue of authenticated key exchange can be defined as follows: We say that a key exchange protocol is authenticated if it provides (either explicit or implicit) key authentication.

The central cryptographic problem in distributed computing practice for authenticated key exchange is how to achieve secure key distribution with both key authentication and confidentiality properties. Of course entity authentication also plays a vital important role for a certain class of AKE protocols to fulfill the requirements of real-world applications.

These complicate security requirements of AKE make it to be not clear at a glance what comprises of an attack on an AKE protocol. A number of distinct types of attacks (as well as a number of less serious weaknesses) have been unveiled against previous protocols. Before beginning to construct or analyze an AKE protocol, it is necessary to identify what attacks a protocol should withstand, and what security attributes are desirable for a protocol to have. Generally speaking, we first identify two types of attack: (i) passive attacks where an attacker attempts to break the security goal of a protocol
by merely observing honest entities carrying out the protocol, and (ii) active attacks where an attacker additionally subverts the communications herself in any possible ways by injecting messages, intercepting messages, replaying messages, altering messages and by even compromising non-trivial information like session keys, long-term or ephemeral private keys. Note that the leakage of session specific critical information can happen in a variety of ways that may include the simple mishandling of information, a temporary break-in into a computer system or the action of a malicious insider. We basically assume that all the secret information stored at a party is potentially vulnerable to any forms of leakage. Obliviously it is essential for any secure protocols to resist with both passive and active attacks since an attacker can be reasonably assumed to have these capabilities over public network.

A number of desirable security attributes for AKE have also been identified so far in which the most prominent ones include resistance to a variety of explicit attacks such as known session key (KSK) attacks, key-compromise impersonation (KCI) attacks, resistance to leakage of session state and the provision of forward secrecy (FS). We informally introduce those attributes as following:

- Resistance to known session key attacks. A protocol providing this attribute should be secure in the face of an attacker who has learned some previous session keys. The resistance to known-key attacks requires that keys from different sessions should be fully computationally independent – namely from learning one session key nothing can be implied about the value of other session keys. This attribute enforces other security properties as well.

- Resistance to key compromise impersonation attacks. In these KCI attacks, firstly studied by Just and Vaudenay [JV96], the attacker learns a party Alice’s long-term private key and then tries to impersonate another honest party Bob to Alice. Resistance to KCI attacks is important in situations where an attacker wishes to obtain some information possessed by Alice, who is only willing to divulge this information to Bob (and where the attacker is not able to obtain Bob’s long-term private key).

- Resistance to leakage of session state. One may want to guarantee that the leakage of session specific state from sessions will have no effects on the security of target sessions. Note that the states that allow to leak depend on specific class of protocols. The disclosed session state might be for instance the exponent of a Diffie-Hellman key or intermediate value generated by certain primitive.

- Provision of forward secrecy. The forward secrecy is an important security property for AKE that was first studied by Günter [Gün90] and reformulated in literature [DvW92] by Diffie et al. A protocol with forward secrecy provides a security
guarantee of sessions in the case that the attacker has learned the long-term private keys of one or more parties. An AKE protocol is said to have the forward secrecy property if the leakage of the long-term key of a party does not compromise the secrecy of previous session keys.

Each attribute may be thought of desirable by AKE protocols. Without loss of generality, we call a protocol as strongly secure AKE protocol if it provides secure key distribution with satisfying all above attributes. However these attributes might differ in detail. For example, for a certain class of AKE, the session states might not be allowed to leak from target sessions (i.e. the test session and its partner session). We will elaborate formal security notions of those attributes in conjunction of the security models presented in Chapter 3. Although there might exist other security properties for specific class of protocols (e.g. the insider security [KS05] for group AKE protocols), we only consider the strongest security requirements that our proposed solutions can provide in this thesis.

1.2 On Security Models for AKE

Early design and evaluation of public AKE protocol have been done merely in a heuristic manner based on trial-and-error. Under this approach lots of practical attacks on those schemes were overlooked. In addition cryptosystems are normally flawed due to different reasons in the real world applications. Therefore the paradigm of provable security is an outcome of these insights. The essential part of the security proofs we consider is a security reduction that takes an algorithm (adversary) for attacking a protocol to break formalized security goals and builds from it an algorithm for solving certain assumed computational problem believed to be hard. This approach was first adopted by famous cryptographic researchers Goldwasser and Micali [GM82, GM84, GMR85] who introduced the framework from computational complexity theory for the purpose of rigorously defining the security of cryptographic schemes. In this section we first provide an overview of existing indistinguishability-based security models for analyzing two party AKE protocols and group AKE protocols respectively. In the latter we briefly outline the security models used in this thesis to prove corresponding proposed solutions. These models might differ in three main aspects: (i) the execution environment, (ii) the adversary model, and (iii) provable security goals.

The execution environment defines general aspects of protocol execution to be independent of protocol specification. For example, the details of session representation and execution may involve general internal state of sessions (e.g. the session keys, transcripts and decision status) and corresponding rules for updating these states.

The adversary model describes the capabilities of any adversary that formalized the general attacks which can be done on considered protocol as illustrated in the previous
section. It always consists of a list oracle queries that the adversary can use to interact with execution environment simulated by AKE challenger.

The security goals (AKE security definitions) define what a protocol should satisfy, which may include the entity authentication and key distribution (session key security). Meanwhile, two important elements may associate with the AKE security definitions are the partnership of two sessions (which might be referred to as matching sessions, matching conversations or partnering in different literatures) and the freshness of target session. Partnership is most fundamental element that may be used to define AKE security in two distinct ways. First, it is used to define the freshness where the latter formally define the restrictions of adversary on performing oracle queries on target session and its partner session. Second, it is used to define the security of entity authentication that a fresh session is required to accept with unique partner session. For an indistinguishability-based AKE model, the security of key exchange is defined to require that the adversary is unable to distinguish the real session key of any fresh session from a random value.

1.2.1 On Models for Two Party AKE

The seminal security notion for two party AKE (2AKE) was provided by Bellare and Rogaway [BR94] who provide the first formal definition for a model of adversary capabilities in association with definition of security (which we refer to as the BR93 model in this thesis). In BR93 model, security goals are formalized by requiring the protocol to provide security of not only key distribution but also entity authentication. Partnership in BR93 is defined using the notion of matching conversations, where a conversation is defined to be the sequence of messages sent and received by an oracle (which emulates a session). In addition the BR93 model captures basic security requirements for AKE such as known session key security and impersonation resilience. However, the BR93 model doesn’t formalize more complicated situations where a long-term private key or session state of a party has been leaked. Since then many variants of BR93 model have been proposed. In 1995, Bellare and Rogaway (BR95) [BR95] presented a formal model for securing delegated AKE protocol where a server (delegate) is involved for distributing session key for two clients. In contrast to original BR93 model, BR95 model introduced a new partner function instead of matching conversations, and in particular a new notion of strong corruption that allows adversary to learn the long-term private key and internal session state of a party (which has not been erased) and even to substitute the victim’s long-term key of her choice. In the latter, Blake-Wilson, Johnson and Menezes [BWJM97] independently proposed model for analyzing security of two-party key exchange (2KE) protocols in the shared and public key settings which use an extension to the BR93 model and combines with the strong corruption from BR95 model. A more recent revision to the model was proposed in 2000 by Bellare et al. [BPR00] hereafter
referred to as the BPR2000 model, in which the session identifier is defined just by requiring to be a unique string generated during the execution of each protocol instance. We note that all above models fail to cover all strong security attributes mentioned in Section 1.1.

During recent years, many new models have been introduced to provide stronger security guarantees than previous works. Meanwhile the most prominent examples are models introduced by Canetti and Krawczyk [CK01] in 2001, and the extended Canetti and Krawczyk model [LLM06, LLM07] introduced by LaMacchia, Lauter and Mityagin in 2007, which are therefore referred to as CK2001 model and eCK07 model respectively. We first remark that, unlike the BR93 model, both CK2001 and eCK07 model only formulate the security for key distribution, i.e. without the entity authentication.

The CK2001 model introduced a new strong query \texttt{StateReveal} query to separately model the leakage of internal secret state of running sessions in contrast to BR95 model. Analysis of a protocol in the CK2001 model requires that the protocol includes unique session identifier which is ‘externally supplied’ by a calling protocol which calls the AKE protocol, and also requires the calling protocol to make sure that two parties that wish to exchange a session key will activate matching sessions. It is unclear how the calling protocol can ensure the second requirement over an insecure network in the presence of an active adversary without an additional security mechanism (so it is possibly unnecessary). It is also impractical for most of AKE protocol to determine the session identifier before session activation. Moreover the CK2001 model does not cover resistance to KCI attacks, and hence protocols proved secure in the model have to be examined on a case-by-case basis for checking KCI resilience. Another shortcoming of the CK2001 model is that it does not account for the security of a target session for which a \texttt{StateReveal} query has been issued.

Krawczyk [Kra05] presented a modification to the CK2001 model when analyzing the security of HMQV protocol [Kra05], which will be referred to as CK\textsubscript{HMQV} model. In contrast to CK2001 model, the partnership in this model is defined via matching sessions that is determined by a special form of the session identifier generated during protocol execution which consists of the participants’ identities concatenated with two transcripts recording messages sent and received by the session owner respectively. In particular, the new partnership definition consider the case about simultaneous initiation of a session by both session participants that is allowed by many two pass symmetric protocols (such as HMQV protocol). The simultaneously protocol execution property is a desirable property in some network settings, and is not covered by previous partnership definition like matching conversations. The CK\textsubscript{HMQV} model introduced a new security notion with respect to weak perfect forward secrecy (wPFS). Whereas wPFS is introduced due to the generic PFS attack [Kra05] to one round AKE protocols.\footnote{We briefly recall the attack as follows. Consider a two-pass AKE protocol in which parties exchange ephemeral public Diffie-Hellman keys, i.e., \(g^x\) and \(g^y\), where \(x\) and \(y\) are chosen at random from}
1 Introduction

PFS security notion, the wPFS implies that an adversary cannot recover a session key with a restriction that the adversary does not modify messages received by the test session and the session is executed before the long-term private keys of participants are compromised. In addition, Krawczyk independently formalized KCI attacks (by allowing the adversary to corrupt the owner of test session) and leakage of session states from target session (and its partner session) in the CK_HMQV model. However, CK_HMQV model does not cover the situations on different combinations of the party corruption and leakage of session state on target session and its partner session. For example, when modeling KCI resilience security property, it does not allow the adversary to learn the session state from the partner session of target session.

In eCK07 model, a new query EphemeralKeyReveal is introduced instead of StateReveal query, which is claimed to cover all ‘session-specific secret’ information and is allowed to be issued to all sessions (including target session and its partner session). The session identifier in eCK07 model is also determined on the fly using a similar approach as CK_HMQV model. However surprisingly the partnership defined via such session identifier fails to cover property of simultaneous protocol execution. We note that the eCK07 satisfies all known security requirements for AKE including resistance to KCI attacks, provision of weak perfect forward secrecy (wPFS), and resistance to LSS from target session and its partner session. Hence it is known to be one of the ‘strongest’ models for AKE. The eCK07 has many variants which may slightly differ in adversary model and partnership definition. For example, Cremers introduced an eCK’ model based on the idea that StateReveal query is stronger than EphemeralKeyReveal. In literature [Oka07], Okamoto used a model which is identical to eCK07 model except for the partnership definition which follows that of CK_HMQV model. In this thesis, We would like to call all variants of eCK07 models as eCK model to avoid ambiguity since they are based on the similar freshness restriction as the original eCK07 model that is different from the CK2001 model.

1.2.2 On Models for Group AKE

Group AKE security models are essentially generalized from security models for 2AKE to the case of multiple parties. Therefore we could also find a lot of common security

\[ \mathbb{Z}_p \text{ (for some large prime p)} \] and \( g \) is a generator of a group with prime order \( p \). The adversary impersonating party Bob could first either generate a random values \( y \in \mathbb{Z}_p \) of her own choice (if possible) or obtain compromised the ephemeral \( Y = g^y \) from a session of Bob (i.e. in any way the exponent \( y \) is known by adversary), and then sends \( Y = g^y \) (together with any other messages) to a session at Alice. The victim Alice responds by sending ephemeral key \( X = g^x \) and computes the session key. The adversary chooses Alice’s session as the test-session, i.e. the session under attack, and reveals Bob’s long-term private key after Bob’s session terminates. Now the adversary can simply follow all protocol steps that an honest party Alice would have performed using ephemeral private key \( x \) and her long-term private key. In particular, the adversary can compute the same session key as target session violating PFS.
requirements among GAKE models and 2AKE models. The formal security model for GAKE protocols was first studied by Bresson et al. [BCPQ01], where the secrecy (indistinguishability) of the established group key and entity authentication are modeled following BR93 model for 2AKE. Since then many modifications and improvements on GAKE models appeared thereafter. The classical GAKE-security notion defined in [BCP01, BCP02, KY03, BMS07], may only provide security against outsider adversary, i.e. assuming that adversary is not part of the group. As pointed out by Katz and Shin [KS05] the early GAKE security notions do not take into account any protection against insider attacks, e.g. preventing honest users from computing different keys and from having distinct views on the identities of other participants. Several models [KS05, BM08, GBN09, GBG09] have been proposed to augment GAKE security against insider threats. Besides consideration of outsider and insider security, formulating stronger security notions for GAKE recently has gained attention of the research community. Quite a few attempts have been made in order to enlarge the class of attacks that a protocol can resist with. Gorantla et al. (GBG) [GBN09] inspired by two party approaches [Kra05] reformulated the key compromise impersonation (KCI) resilience attribute for GAKE by considering outsider and insider KCI attacks respectively. The GBG model [GBN09] also considered the leakage of secret states as in two party CK2001 model [CK01] that allows the adversary to obtain long-term private keys and secret states independently, with a restriction that the leakage of secret states from sessions for which the adversary is not required to distinguish the key. Such restriction is quite necessary since many GAKE protocols are insecure if secret session states used to compute session key are exposed.

Most recently, Fujioka et al. [FMSU12] used a model which is basically generalized from the two-party eCK model to group case (we will refer to this model as g-eCK model), to prove a state-of-art tripartite one-round key exchange protocol. Analogous to eCK model, the g-eCK model satisfies required security properties aforementioned for AKE. Thus the g-eCK model is also known to be one of the strongest models for proving GAKE protocols.

### 1.2.3 Selecting Appropriate Models for AKE Protocols

In real-world applications, we always have different requirements on new constructions for AKE that may yields distinct types of AKE protocols. In this thesis, we mainly focus on two classes of AKE protocols: (i) generic AKE compilers, that construct new AKE protocols based on existing system (e.g. passive secure key exchange) without any modification on it, and (ii) one-round AKE protocols (for either two party or group cases). Thus we briefly describe our choice of the formal models.

Our strategy of selecting appropriate model follows the similar approach in [JKSS12, Appendix A]. The main guideline for our choice is that the chosen model for a consid-
erated protocol should cover attacks as much as possible which the protocol can resist with. As for AKE compilers, we would like to first choose a model formalizing entity authentication which is not covered by recently models such as eCK model or CK2001 model. Intuitively, protocols providing security for both entity authentication and key distribution are ‘stronger’ than protocols that can only provide key distribution security. On the other hand, the eCK model and CK_HMQV are ‘too strong’ for AKE compilers, since these models allow the adversary to reveal session states from target session and its partner session that would result in the protocol being trivially broken. Whereas the CK2001 model neither covers all necessary security properties nor security goal about entity authentication that is provided by AKE compilers. In addition, the eCK model doesn’t model perfect forward secrecy in contrast to CK2001 model. Therefore, our approach is to somewhat strengthen the CK2001 model by allowing adversary to corrupt the target session to model KCI attacks and formally define the security for entity authentication, that yield a modified CK (mCK) model.

However for a specific class of AKE protocol which can only provide key distribution security, e.g. one-round key exchange protocols (with implicit key authentication) are unable to guarantee that two sessions are partnered within two protocol moves due to the asymmetric of the communication. For this reason, the eCK model and g-eCK model are suitable to our constructions for secure one-round AKE protocols. However we are not going to use the session identifier of eCK07 model, since it is unable to model simultaneous protocol execution that might be a very important attribute of one-round protocol. Instead we will adopt a general approach to define the session identifier for mCK, eCK and g-eCK models used by our proposals, namely the session identifier is only defined as a protocol specified unique string. The intuition behind this definition strategy is to keep the distinct properties (e.g. simultaneous protocol execution) of each protocol and to circumvent some trivial theoretical attack (e.g. the attack presented in Section 4.5).

1.2.4 Connection Between Practical Implementation and Theoretical Security

Please note that in current models where leakage of session state are modeled leave out the definition of session state or ephemeral key to specific protocols. Since, it is hard to define the session state in a general approach that is independent of any protocols and corresponding implementation mechanisms. However the ambiguities on session state may yield a lot of potential problems either in the construction of a protocol or in its security analysis under those strong models. If any implementer realized a specific AKE protocol in a careless way allowing it to leak non-trivial session state to attackers, then it would trivially invalidate the security proof in those strong models. On the other hand, to our best of knowledge, no AKE protocol is secure in the (e)CK model if ‘all’ session
states can be revealed. Namely some session states of AKE protocols should be leakage resilience. Those critical session states might be, for instances, the pre-image of session key in HMQV protocol [Kra05] and the decapsulation key of KEM in Boyd et al.’s scheme [BCNP08]. In order to protect those critical session states of AKE protocols, utilizing secure (tamper-proof) device might be a natural solution. In this way it is possible to adopt a “All-and-Nothing” strategy to define the session state – namely we can assume that all states stored on untrusted host machine (e.g. personal computer) can be revealed via \texttt{StateReveal} query and no state would be exposed at secure device. Of course one could distribute all protocol computations on the secure device then the security model would equal to a model without \texttt{StateReveal} query. However the security result of a protocol analysed under such implementation scenario must be weaker than that in a case allowing leakage of session states. In contrast, our goal is to define the maximum states that can be leaked. As those secure devices might be short in both storage capacity and computational resource, the algorithm on secure device normally causes performance bottleneck of systems. This also makes it necessary to optimize AKE protocols when they are realized involving secure device.

1.2.5 Constructions in the Standard Model

In this thesis we focus on AKE constructions which are secure the in standard model (also known as bare model and plain model). In the standard model, the resources of adversary are bounded polynomially by the amount of time and computational power available. Cryptographic protocol that can be proven secure based on only complexity assumptions (which may correspond to some hard problem, such as Bilinear Decisional Diffie-Hellman problem, cannot be solved in probabilistic polynomial time) is said to be secure in the standard model. However, it is notoriously difficult to do security reductions in the standard model, so that many proofs are given in alternative models where certain cryptographic primitive is replaced with its idealized form. The most prominent example of this technique is the random oracle model that is first introduced by Bellare and Rogaway [BR93] as a trade-off between provable security and practical efficiency. In the random oracle model, all parties including the adversary have black-box access to functions (i.e. the random oracles) which behave like truly random functions. However in practical implementations, corresponding ideal random functions (oracles) are realized by concrete objects such as cryptographic hash functions.

Unfortunately the security proof in the random oracle may not imply that corresponding protocol is secure in the real world. At most, it guarantees the security from a certain class of attacks that treat the hash function as a black-box, rather than the security in general. Several results, e.g., [CGH98, BBP04], have demonstrated that there exist schemes which are provably secure in the random oracle model, but are insecure as soon as one replaces the random oracle by any concrete hash functions. These results
imply that a security reduction in the random oracle model is not always useful. This also makes the schemes secure in the standard model to be more appealing than that in the random oracle model.

1.3 Structure of This Thesis

In this thesis we deal with the novel construction for different kinds of AKE protocols which are provably secure in strong models without random oracles. We recall basic security definitions of important primitives in Chapter 2 that are used in our constructions. Most of these definitions are commonly used in previous literatures, except for a small part of them which are derived from well-known assumptions (e.g. the Cube Bilinear Decisional Diffie-Hellman (CBDDH) assumption that is modified from the standard Bilinear Decisional Diffie-Hellman (BDDH) assumption [Jou00]).

In Chapter 3, we introduce different models to formulate the security for our specific proposed AKE protocols respectively. We first define a general execution environment in Section 3.1 and also a general adversary model in Section 3.2. On the next, two security definitions (i.e. mCK security and eCK security) for two party AKE protocols are defined in Section 3.3 and the group AKE security definition (i.e. g-eCK security) is given in Section 3.4.

In Chapter 4 we are going to present three 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 of proposed compilers is shown in the mCK model without random oracles. Those results are joint works with Yong Li and Sven Schaege that originate from the paper in submission: New Modular Compilers for Authenticated Key Exchange. We also illustrate a new theoretical attack related to partnership definition in Section 4.5 which is discovered by me.

We show constructions for one-round two party AKE in Chapter 5. First of all, a generic construction is presented in Section 5.2. On the second, a concrete instantiation of such generic construction is given in Section 5.3 which aims to improve the efficiency. The resultant protocols are proven to be eCK secure in the standard model. These results originate from [Yan13a, Yan13b].

In Chapter 6 we solve the problem regarding constructions for g-eCK secure one-round group AKE protocols. Basically, we first generalize the construction of two party protocol in Section 5.3 to three party case in Section 6.2. In the later, we extend the one-round three party AKE protocol to a construction for group AKE in Section 6.3. Meanwhile, we show how to simplify the proof for one-round GAKE protocols under the g-eCK model in Section 6.1.5. These results come from a paper with Yong Li that is currently in submission [LY13].
2 Preliminaries

In this chapter, we recall the required definitions for proposed protocols.

Notations. We let \( \kappa \in \mathbb{N} \) denote the security parameter and \( 1^\kappa \) the string that consists of \( \kappa \) ones. Let a ‘hat’ on top of a capital letter denote an identity; without the hat the letter denotes the public key of that party. Let \( [n] = \{1, \ldots, n\} \subset \mathbb{N} \) be the set of integers between 1 and \( n \). If \( S \) is a set, then \( a \overset{\mathcal{R}}{\leftarrow} S \) denotes the action of sampling a uniformly random element from \( S \). Let ‘\( || \)’ denote the operation concatenating two binary strings. Let \( K_{\text{AKE}} \) be the key space of session key, and \( \{\mathcal{PK}, \mathcal{SK}\} \) be key spaces for long-term public/private key respectively. Let \( I\mathcal{DS} \) be an identity space. Those spaces are associated with security parameter \( \kappa \) of considered protocol.

Paper. In the sequel, most of the definitions are common used in previous literatures, except for partial results from our papers in submission. The KE definition and Lemma 1 in Section 2.2 was formulated by Li, Schaege and I, and the improved notion of tag-based authentication schemes in Section 2.12 was formulated by me in paper (in submission): New Modular Compiler for Authenticated Key Exchange. I defined the Cube Bilinear Decisional Diffie-Hellman assumption in Section 2.16.2 and the n-Multilinear Decisional Diffie-Hellman Assumption in Section 2.16.3 which are used in our proposals.

2.1 Negligible Functions

In this section, we recall the negligible function defined by Oded Goldreich [Gol01] in the following.

Definition 1. We call a function \( \epsilon : \mathbb{N} \to \mathbb{R} \) is negligible if for every positive polynomial \( p(\cdot) \) there exists an \( N \in \mathbb{N} \) such that for all \( n > N \),

\[
\epsilon(n) < \frac{1}{p(n)}.
\]

That is, for all sufficiently large \( n \), a negligible function \( \epsilon(n) \) approaches zero faster than the reciprocal of any polynomial \( p(n) \).
2 Preliminaries

2.2 Key Exchange Protocols

A two party key-exchange (KE) protocol without long-term keys 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 should 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 $l$-move key exchange protocols as sketched below.

KE = (KE.Setup, KE.EKGen, KE.SKGen) consists of three PPT algorithms which may be called by a party $ID \in I\mathcal{D}\mathcal{S}$ in each session. Let $T$ be the transcript of all messages exchanged in a KE protocol instance (see Figure 2.1).

- $pms^{ke} \leftarrow$ KE.Setup($1^{\kappa}$): this algorithm takes as input the security parameter $1^{\kappa}$ and outputs a set of system parameters $pms^{ke}$, e.g. a large prime and a group generator. The parameters $pms^{ke}$ might be implicitly used by other algorithms for simplicity.

- $(esk_{ID}, epk_{ID}, m_{ID}) \leftarrow$ KE.EKGen($pms^{ke}, in$): The PPT ephemeral key generator takes as input the system parameters $pms^{ke}$ and message $in \in \mathcal{M}_{KE}$, and outputs an ephemeral key pair $(esk_{ID}, epk_{ID})$ for the caller $ID$ that consist of the ephemeral secret key $esk_{ID} \in \mathcal{ESK}$ and the ephemeral public key $epk_{ID} \in \mathcal{EPK}$ relative to $esk_{ID}$, and a message $m_{ID} \in \mathcal{M}_{KE}$ that requires to be sent in a protocol move, where $\mathcal{M}_{KE}$ is the message space of KE, $\mathcal{ESK}$ is the space for ephemeral secret key and $\mathcal{EPK}$ is the space for ephemeral public key. A key exchange protocol. The execution of this algorithm might be determined by input message in which could be any information including for examples identities of session participants, ephemeral public key or just empty string $\emptyset$. We assume that the output message $m_{ID}$ should at least include the generated ephemeral public key, i.e. $epk_{ID} \in m_{ID}$. If $epk_{ID} = m_{ID}$, for simplicity we may write $(esk_{ID}, epk_{ID}) \leftarrow$ KE.EKGen($pms^{ke}, in$) for $(esk_{ID}, epk_{ID}, m_{ID}) \leftarrow$ KE.EKGen($pms^{ke}, in$).

- $k \leftarrow$ KE.SKGen($esk_{ID}, T$): 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 a session, and outputs the session key $k$.

We only consider key exchange protocols with perfect correctness that is on input the
same protocol transcript $T$

$$
Pr \begin{bmatrix}
\text{KE.SKGen}(esk_{ID_1}, T) = \text{KE.SKGen}(esk_{ID_2}, T); \\
(esk_I, epk_I, m_{ID_1}) \leftarrow \text{KE.EKGen}(pms^{ke}, in_1), \\
(esk_{ID_2}, epk_R, m_{ID_2}) \leftarrow \text{KE.EKGen}(pms^{ke}, in_2), \\
(m_{ID_1}, m_{ID_2}) \in T.
\end{bmatrix} = 1.
$$

The Figures 2.1 briefly illustrates the general protocol execution of two-move KE.

![Diagram of General Two-move KE Protocol](image)

**Figure 2.1:** General Two-move KE Protocol

**Definition 2.** We say a key exchange protocol without long-term keys is correct if for any protocol instance with session key generated as $k := \text{KE.SKGen}(esk_{ID}, T)$ it holds that $esk_{ID}$ is generated freshly by $\text{KE.EKGen}$ in corresponding protocol instance. That is, each party computes each session key using only ephemeral secret key which is freshly generated by $\text{KE.EKGen}$ in corresponding protocol instance.

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) [DH76], the $\text{KE.EKGen}$ is executed without any additional message i.e. $in_1 = in_2 = \emptyset$, and the generated messages such that $m_{ID_1} = epk_{ID_1}$ and $m_{ID_2} = epk_{ID_2}$. 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 2.1 as: $in_1 = \emptyset$, $in_2 = m_{ID_1} = epk_{ID_1}$. We stress that the key pairs $(esk_{ID_1}, epk_{ID_1})$ and $(esk_{ID_2}, epk_{ID_2})$ may have distinct forms depending on specific KE protocol, which are also determined by the forms of messages $(in_1, in_2)$ while running $\text{KE.EKGen}$. 

In case both parties ‘contribute’ values which are used to computed the session key, i.e. $k \neq esk_{ID_2}$ and $k \neq esk_{ID_1}$, this is very obvious as the contribution of $ID_1$ has to be transmitted to $ID_2$ and vice versa. If only one party $ID_c \in \{ID_1, ID_2\}$ decides on the session key $esk_{ID_c} = k$, then $k$ has to securely be transferred to the other party $ID_c$.
via some form of encryption of \( k \) which is sent in \( epk_{ID_c} \). However, to guarantee that only the single party \( ID'_c \) can decrypt the session key from \( epk_{ID_c} \) 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 \( epk_{ID'_c} \) in the first move of the key exchange protocol. We thus have that \( ID_c = ID_1 \) and \( ID'_c = ID_2 \).

In order to model passive attacks we define an \( \text{Execute}(ID_1, ID_2) \) query. The adversary can use this 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 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 \( \text{Execute}(ID_1, ID_2) \) query, an instance of KE protocol is executed between party \( ID_1 \) and party \( ID_2 \). After simulation this query returns the transcript \( T \) of all messages exchanged in corresponding protocol instance and a session key \( K_0 \).

**Definition 3.** We say that a correct key-exchange protocol \( KE \) is \((t, \epsilon_{KE})\)-passively-secure if for all probabilistic polynomial-time \( t \) adversary holds that \( |\text{EXP}^{ps}_{KE,A}(\kappa) = 1| - 1/2| \leq \epsilon_{KE} \) for some negligible function \( \epsilon_{KE} = \epsilon_{KE}(\kappa) \) in the security parameter \( \kappa \) in the following experiment.

\( \text{EXP}^{ps}_{KE,A}(\kappa) \): On input security parameter \( 1^\kappa \), the security experiment is proceeded as a game between a challenger \( C \) and an adversary \( A \) based on a key exchange protocol \( KE \), where the following steps are performed:

1. A challenger generates a set of identities \( \{ID_1, \ldots, ID_\ell\} \) for potential protocol participants where \( \ell \in \mathbb{N} \). The adversary is given all identities as input and is allowed to interact with challenger via making \( \text{Execute}(ID_i, ID_j) \) query at most \( d \) times for each party where \( d \in \mathbb{N} \) and \( i, j \in [\ell] \). As response, the challenger returns \((T, K_0)\) to adversary.

2. At some point, the adversary outputs a special symbol \( \top \) for challenge. Given \( \top \), the challenger runs a new protocol instance, obtaining the transcript \( T^* \) and key \( K_0^* \), samples \( K_i^* \) uniformly at random from the key space of the protocol, and tosses a fair coin \( b \in \{0, 1\} \). Then it returns \((T^*, K_0^*, b)\) to the adversary. After that the adversary may continue making \( \text{Execute}(ID_i, ID_j) \) queries. Finally, adversary \( A \) may terminate with outputting a bit \( b' \).

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 \( KE.\text{EKGen} \) there cannot be any collisions among the ephemeral public keys generated by certain type of \( KE.\text{EKGen} \). This lemma will be useful in the security proofs of our compilers to show that a compiler does not have to
2 Preliminaries

exchange additional random values after the KE run to guarantee that the transcripts which are authenticated with the authentication mechanism are unique.

Please note that for a two-move and two-party (ID₁ and ID₂) KE protocol there exit at most two types of KE.EKGen algorithm which may be determined by input messages in₁ and in₂. We here explicitly classify the algorithm KE.EKGen into two types denoted by KE.EKGenID₁ for party ID₁ and KE.EKGenID₂ for party ID₂. Of course we may have KE.EKGenID₁ = KE.EKGenID₂ if those two ephemeral key generators are the same type. With respect to different types of KE.EKGen we assume they have different ephemeral key spaces (one could also configure the KE.EKGen to realize this).

This implies the ephemeral keys from different spaces are distinct and of course there is no collision among them. While considering the collisions among ephemeral keys, we let Coll denote the event that: After a polynomial number q times execution of KE.EKGen algorithm there exists two ephemeral public keys epk and epk′ generated by the same type ephemeral key generator KE.EKGen at different time are identical, where the number q is determined by time t.

**Definition 4.** We say all ephemeral keys generated by KE.EKGen are (q, t, ϵcoll) -distinct if those ephemeral keys are generated by KE.EKGen after q times execution of KE.EKGen algorithm within time t and there exists no collision among those ephemeral keys except for probability ϵcoll.

**Lemma 1.** Assume KE is a (t, ϵKE)-passively secure protocol without long-term key as defined above. Then all ephemeral public keys generated by KE.EKGen in the runs of KE are (q, t, ϵcoll)-distinct such that ϵcoll ≤ q · ϵKE.

**Proof.** We first consider the case that KE.EKGenID₁ ≠ KE.EKGenID₂. In this case, there is no collision between ephemeral keys epkID₁ and epkID₂, because those keys are assumed to be generated from different key spaces. So that we need to evaluate the collision probability among ephemeral keys generated by the same type ephemeral key generators. Assume that with non-negligible probability ϵcoll there will be a collision among the epkID₁ after q protocol runs, or a collision among the epkID₂ after q protocol runs. According to the protocol specification the epkID₁ values are computed by randomized runs of KE.EKGenID₁ while the epkID₂ values have been computed by randomized runs of KE.EKGenID₂. In particular, the computation of the epkID₁ and epkID₂ are deterministic in system parameters pmske, message in₁ (resp. in₂) and the internal random coins ωID₁ used by ID₁ and ωID₂ used by ID₂.

Let epk*ID₁ and epk*ID₂ be the ephemeral public keys that are exchanged in the test session and given, together with the challenge key k′ and transcript T′, to the adversary. Let eskID₁ and eskID₂ be the corresponding ephemeral secret keys. These keys have also been computed using KE.EKGen₁ (resp. KE.EKGen₂) with random coin ωID₁ (resp. ωID₂) and in₁ (resp. in₂). The adversary first guesses whether the collision occurs among the
The above lemma can easily be extended to KE.EKGen\textsubscript{ID\textsubscript{1}}, q − 1 times with randomness \(\omega_{ID\textsubscript{1},t}\) and \(i_{n,1}\) to obtain \(\{esk_{ID\textsubscript{1},t}, epk_{ID\textsubscript{1},t}\}\) for \(i \in [1; q - 1]\) in time less than \(t\). With the same probability \(\epsilon_{col}\) it obtains two values \(epk'_{ID\textsubscript{1}}, epk''_{ID\textsubscript{1}}\) among the \(q\) values \(epk^*_{ID\textsubscript{1}}, epk^*_{ID\textsubscript{1},1}, \ldots, epk^*_{ID\textsubscript{1},q-1}\) with \(epk'_{ID\textsubscript{1}} = epk''_{ID\textsubscript{1}}\). Since it holds with probability \(\geq 2/q\) that either \(epk'_{ID\textsubscript{1}} = epk^*_{ID\textsubscript{1}}\) or \(epk''_{ID\textsubscript{1}} = epk^*_{ID\textsubscript{1}}\). In this case the adversary knows one pair \((\omega'_{ID\textsubscript{1},t}, i'_{n,1})\) which deterministically maps to \((esk^*_{ID\textsubscript{1}}, epk^*_{ID\textsubscript{1}})\). It enables \(A\) to break the passive security with advantage at least \(\frac{\epsilon_{col}}{q}\). Since the secret \(esk^*_{ID\textsubscript{1}}\) is the only secret that is required to compute the final session key where all messages in \(T^*\) are publicly known. Next the adversary can compare whether \(k = k'\) and correctly guess \(b'\). In cases that \(KE.\)EKGen\textsubscript{ID\textsubscript{1}} \(\neq\) KE.\)EKGen\textsubscript{ID\textsubscript{2}} and there is a collision among the \(epk_{ID\textsubscript{2}}\) the situation is similar. The adversary runs KE.\)EKGen\textsubscript{ID\textsubscript{2}}, q − 1 times with fresh randomness \(\omega_{ID\textsubscript{2},t}\) on each invocation. Since all the \(\omega_{ID\textsubscript{2},t}\) have been drawn uniformly random, the adversary may have after \(q - 1\) honest executions of the protocol computed two values \(epk'_{ID\textsubscript{2}}, epk''_{ID\textsubscript{2}}\) among the \(q\) values \(epk^*_{ID\textsubscript{2}}, epk^*_{ID\textsubscript{2},1}, \ldots, epk^*_{ID\textsubscript{2},q-1}\) such that \(epk'_{ID\textsubscript{2}} = epk''_{ID\textsubscript{2}}\). With probability \(\geq 2/q\) we have either \(epk_{ID\textsubscript{2}} = epk^*_{ID\textsubscript{2}}\) or \(epk_{ID\textsubscript{2}} = epk^*_{ID\textsubscript{2}}\). Similar to before, the adversary then knows the corresponding \(esk'_{ID\textsubscript{2}} = esk^*_{ID\textsubscript{2}}\) and thus also KE.SKGen\textsubscript{ID\textsubscript{2}}(\(esk^*_{ID\textsubscript{2}}, T^*\)) = \(k^*\). Furthermore, if \(KE.\)EKGen\textsubscript{ID\textsubscript{1}} = KE.\)EKGen\textsubscript{ID\textsubscript{2}} the result is directly implied by the above proof based on KE.\)EKGen\textsubscript{ID\textsubscript{1}}. Hence, due to the security of KE protocol, we have that the probability bound \(\frac{\epsilon_{col}}{q} \leq \epsilon_{KE}\).

\[\square\]

### 2.2.1 General KE Protocols

The above definition of KE, the corresponding security definition, and the results of the above lemma can easily be extended to \(l\)-move KE protocols in the natural way. Concretely, besides the KE.Setup and KE.SKGen algorithms, each party may run at most \([l/2]\) different types of KE.EKGen algorithm in each protocol instance depending on the input messages \(i\subscript{n,i} : 1 \leq i \leq l\). Namely, each session participant can call at most \([l/2]\) of times of KE.EKGen algorithms during protocol execution. We let each invocation of algorithm KE.EKGen in \(i\)-move (\(1 \leq i \leq l\)) as KE.EKGen\textsubscript{i} which is used to compute the message for \(i\)-move, where the odd numbered moves are called by ID\textsubscript{1} and the even numbered moves are invoked by ID\textsubscript{2}. Consequently, we may have (for instance when \(l\) is even) a series of executions:

\[
\begin{align*}
(esk_{ID\textsubscript{1},1}, epk_{ID\textsubscript{1},1}, m_{ID\textsubscript{1},1}) & \overset{\text{i}}{\rightarrow} KE.\)EKGen\textsubscript{1}(\(pms\textsuperscript{ke}, i\subscript{n,1}\)), \\
(esk_{ID\textsubscript{2},2}, epk_{ID\textsubscript{2},2}, m_{ID\textsubscript{2},2}) & \overset{\text{i}}{\rightarrow} KE.\)EKGen\textsubscript{2}(\(pms\textsuperscript{ke}, i\subscript{n,2}\)), \ldots, \\
(esk_{ID\textsubscript{1},l-1}, epk_{ID\textsubscript{1},l-1}, m_{ID\textsubscript{1},l-1}) & \overset{\text{i}}{\rightarrow} KE.\)EKGen\textsubscript{l-1}(\(pms\textsuperscript{ke}, i\subscript{n,l-1}\)), \\
(esk_{ID\textsubscript{2},l}, epk_{ID\textsubscript{2},l}, m_{ID\textsubscript{2},l}) & \overset{\text{i}}{\rightarrow} KE.\)EKGen\textsubscript{l}(\(pms\textsuperscript{ke}, i\subscript{n,l}\)).
\end{align*}
\]

In particular one could think that there exists a general ephemeral key generator

17
KE.EKGen_{ID_1}, for party ID_1 that combines all those invocations KE.EKGen_{ID_1}, KE.EKGen_{ID_2}, \ldots, KE.EKGen_{ID_{2i}} for 1 \leq i \leq \lceil l/2 \rceil. Namely KE.EKGen_{ID_1} is executed as (esk_{ID_1}, epk_{ID_1}, m_{ID_1}) \xleftarrow{\$} KE.EKGen_{ID_1}(pms^{sk}, in_{ID_1}) where 
\begin{align*}
in_{ID_1} & = (in_1, in_3, \ldots, in_{2i-1}, in_{2i+1}), \\
esk_{ID_1} & = (esk_{ID_1,1}, esk_{ID_1,3}, \ldots, esk_{ID_1,2i-1}, esk_{ID_1,2i+1}), \\
epk_{ID_1} & = (epk_{ID_1,1}, epk_{ID_1,3}, \ldots, epk_{ID_1,2i-1}, epk_{ID_1,2i+1}) \\
m_{ID_1} & = (m_{ID_1,1}, m_{ID_1,3}, \ldots, m_{ID_1,2i-1}, m_{ID_1,2i+1}).
\end{align*}

Similarly, we could have the general ephemeral key generator KE.EKGen_{ID_2} for party ID_2. We could therefore apply the result of Lemma 1 to the general KE, namely with overwhelming probability there is for instance no collision among all epk_{ID_1} generated by KE.EKGen_{ID_1}.

### 2.3 Digital Signature Schemes

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

- \((sk, pk) \xleftarrow{\$} \text{SIG.Gen}(1^\kappa)\): this algorithm takes as input the security parameter \kappa and outputs a (public) verification key \(pk \in PK\) and a secret signing key \(sk \in SK\);
- \(\sigma \xleftarrow{\$} \text{SIG.Sign}(sk, m)\): the signing algorithm generates a signature \(\sigma \in \mathcal{S}_{SIG}\) for message \(m \in \mathcal{M}_{SIG}\) using signing key \(sk\);
- \(\{0, 1\} \leftarrow \text{SIG.Vfy}(pk, m, \sigma)\): on input verification key \(pk\), a message \(m \in \mathcal{M}_{SIG}\) and corresponding signature \(\sigma\), the verification algorithm outputs 1 if \(\sigma\) is a valid signature for \(m\) under key \(pk\), and 0 otherwise.

We review the standard notion of security for signature schemes introduced by Goldwasser, Micali and Rivest [GMR88].

**Definition 5.** We say that SIG = (SIG.Gen, SIG.Sign, SIG.Vfy) is \((q, t, \epsilon_{SIG})\)-secure against existential forgeries under adaptive chosen-message attacks, if \(\Pr[\text{EXP}^{\text{euf-cma}}_{SIG,A}(\kappa) = 1] \leq \epsilon_{SIG}\) for all adversaries \(A\) running in time at most \(t\) in the following experiment:

\[
\begin{align*}
(\sigma^*, m^*) & \leftarrow A^{SIG(sk, \cdot)} & \\
\text{return 1, if the following conditions are held:} & \\
1. & \text{SIG.Vfy}(pk, m^*, \sigma^*) = 1, \text{ and} \\
2. & \text{m^* is not submitted to SIG}(sk, m^*) oracle; 
\end{align*}
\]
output 0, otherwise;

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

2.4 Public Key Encryption Schemes

A PKE scheme consists of three polynomial time algorithms

$PKE = (PKE.\text{KGen}, PKE.\text{KGen}, PKE.\text{Enc}, PKE.\text{Dec})$ with the following semantics:

- $pms^{\text{pke}} \leftarrow PKE.\text{Setup}(1^\kappa)$: this algorithm takes as input a security parameter $1^\kappa$ and outputs a set of system parameters $pms^{\text{pke}}$. The parameters $pms^{\text{pke}}$ might be implicitly used by other algorithms for simplicity.

- $(pk, sk) \leftarrow PKE.\text{KGen}(pms^{\text{pke}})$: this algorithm takes as input the system parameter $pms^{\text{pke}}$, and outputs a pair of encryption/decryption keys $(pk, sk) \in \{PK, SK\}$;

- $C \leftarrow PKE.\text{Enc}(pk, m)$: this algorithm takes as input takes as input a public key $pk$ and a message $m \in M_{\text{PKE}}$, and outputs ciphertext $C \in C_{\text{PKE}}$, where $M_{\text{PKE}}$ is a message space and $C_{\text{PKE}}$ is a ciphertext space.

- $m \leftarrow PKE.\text{Dec}(sk, C)$: this algorithm takes as input a key $sk$ and a ciphertext $C \in C_{\text{PKE}}$, and outputs either a message $m \in M_{\text{PKE}}$ or an error symbol $\bot$.

We recall the Rackoff and Simon’s definition [RS92] that is widely used to provide security argument for public key encryption scheme, i.e. the notion of security against adaptive chosen ciphertext attacks.

**Definition 6.** We say that a PKE scheme $PKE = (PKE.\text{Setup}, PKE.\text{KGen}, PKE.\text{Enc}, PKE.\text{Dec})$ is $(q, t, \epsilon_{PKE})$-secure (message indistinguishable) against adaptive chosen-ciphertext attacks, if the probability bound $|\Pr[\text{EXP}_{\text{PKE}, A}^{\text{ind-cca}}(\kappa) = 1] - 1/2| \leq \epsilon_{PKE}$ holds for all probabilistic polynomial-time (PPT) adversaries $A$ that make a polynomial number of oracle queries $q$ while running in time at most $t$ in the following experiment.

$\text{EXP}_{\text{PKE}, A}^{\text{ind-cca}}(\kappa)$

$pms^{\text{pke}} \leftarrow PKE.\text{Setup}(1^\kappa), (pk, sk) \leftarrow PKE.\text{KGen}(pms^{\text{pke}})$;

$(m_0, m_1, st) \leftarrow A^{\text{DEC}(sk, \cdot)}(pk), (m_0, m_1) \in M_{\text{PKE}}$;

$b \leftarrow \{0, 1\}, (C^*) \leftarrow PKE.\text{Enc}(pk, m_b)$;

$b' \leftarrow A^{\text{DEC}(sk, \cdot)}(C^*, st)$;

if $b = b'$ then return 1, otherwise return 0;

where $\epsilon_{\text{PKE}} = \epsilon_{\text{PKE}}(\kappa)$ is a negligible function in $\kappa$, and on input $C$ the oracle $\text{DEC}(sk, C)$ returns $m \leftarrow PKE.\text{Dec}(sk, C)$ with the restriction that $A$ is not allowed to query $\text{DEC}(sk, \cdot)$ on the challenge ciphertext $C^*$. 
2 Preliminaries

2.5 Key Encapsulation Mechanism

In our third compiler we can use a key encapsulation mechanism. We stress that we could also use a IND-CCA secure public key encryption (PKE) system by first generating a random session key and then encrypting this key with the encryption system. However, PKEs in general are less efficient than KEMs as they have to provide security for arbitrary plaintext distributions whereas a KEM only has to work for randomly distributed plaintexts.\footnote{It is well-known that KEMs do give rise to PKEs via the KEM/DEM paradigm.} A KEM scheme consists of four polynomial time algorithms $\text{KEM} = (\text{KEM.Setup, KEM.KGen, KEM.EnCap, KEM.DeCap})$ with the following semantics:

- $\text{KEM.Setup}(1^\kappa)$: this algorithm takes as input a security parameter $1^\kappa$ and outputs a set of system parameters $\text{pms}^{\text{kem}}$. The parameters $\text{pms}^{\text{kem}}$ might be implicitly used by other algorithms for simplicity.
- $(pk,sk) \leftarrow \text{KEM.KGen}(\text{pms}^{\text{kem}})$: a key generation algorithm which on input system parameter $\text{pms}^{\text{pke}}$ and outputs a pair of encryption/decryption keys $(pk,sk)$;
- $(K,C) \leftarrow \text{KEM.EnCap}(pk)$: an encryption algorithm which takes as inputs an encryption key $pk$, outputs a session key $K \in \mathcal{K}_{\text{KEM}}$ and ciphertext $C \in \mathcal{C}_{\text{KEM}}$, where $\mathcal{K}_{\text{KEM}}$ is a session key space and $\mathcal{C}_{\text{KEM}}$ is a ciphertext space.
- $(K) \leftarrow \text{KEM.DeCap}(sk,C)$: a decryption algorithm which takes as input a decryption key $sk$ and a ciphertext $C \in \mathcal{C}_{\text{KEM}}$, and outputs a session key $K \in \mathcal{K}_{\text{KEM}}$.

All ‘spaces’ for the corresponding values are parametrized with security parameter $\kappa$.

Definition 7. We say that a key encapsulation mechanism scheme $\text{KEM} = (\text{KEM.Setup, KEM.KGen, KEM.EnCap, KEM.DeCap})$ is $(q,t,\epsilon_{\text{KEM}})$-secure (key indistinguishable) against adaptive chosen-ciphertext attacks, if the probability bound

$$|\Pr[\text{EXP}^{\text{ind-cca}}_{\text{KEM,A}}(\kappa) = 1] - 1/2| \leq \epsilon_{\text{KEM}}$$

for all adversaries $A$ running in time at most $t$ in the following experiment.

\begin{align*}
\text{EXP}^{\text{ind-cca}}_{\text{KEM,A}}(\kappa) \\
\text{pms}^{\text{kem}} \leftarrow \text{KEM.Setup}(1^\kappa), (pk,sk) \leftarrow \text{KEM.KGen}(\text{pms}^{\text{kem}}); \\
st \leftarrow A^{\mathcal{D}\mathcal{E}\mathcal{C}(sk,\cdot)}(pk), \text{which can make up to } q \text{ queries} \\
to \text{oracle } \mathcal{D}\mathcal{E}\mathcal{C}(sk,\cdot); \\
(K^*_0,C^*) \leftarrow \text{KEM.EnCap}(pk), K^*_1 \leftarrow \mathcal{K}_{\text{KEM}}, b \xleftarrow{} \{0,1\}; \\
b' \leftarrow A^{\mathcal{D}\mathcal{E}\mathcal{C}(sk,\cdot)}(pk,K^*_b,C^*,st); \\
\text{if } b = b' \text{ then return } 1, \text{ otherwise return } 0;
\end{align*}

where $\epsilon_{\text{KEM}} = \epsilon_{\text{KEM}}(\kappa)$ is a negligible function in security parameter $\kappa$, and on input $C$ the oracle $\mathcal{D}\mathcal{E}\mathcal{C}(sk,\cdot)$ returns $K \leftarrow \text{KEM.DeCap}(sk,C)$ with the restriction that $A$ is not allowed to query $\mathcal{D}\mathcal{E}\mathcal{C}(sk,\cdot)$ on the challenge ciphertext $C^*$.\footnote{It is well-known that KEMs do give rise to PKEs via the KEM/DEM paradigm.}
2 Preliminaries

2.6 Non-Interactive Key Exchange Protocols

We consider a Non-Interactive Key Exchange (NIKE) scheme (following [FHKP13]) in the public key setting consists of three algorithms: NIKE.Setup, NIKE.KGen and ShareKey associated with an identity space $\mathcal{I}DS$ and a shared key space $\mathcal{K}_{\text{NIKE}}$, in which those algorithms have following semantics:

- $pms^{\text{ni}}ke \leftarrow \text{NIKE.Setup}(1^\kappa)$: this algorithm takes as input a security parameter $1^\kappa$ and outputs a set of system parameters $pms^{\text{ni}}ke$. The parameters $pms^{\text{ni}}ke$ might be implicitly used by other algorithms for simplicity.

- $(sk_{ID}, pk_{ID}, pf_{ID}) \leftarrow \text{NIKE.KGen}(pms^{\text{ni}}ke, ID)$: this algorithm takes as input parameters $pms^{\text{ni}}ke$ and an identity ID, and outputs a pair of long-term private/public key $(sk_{ID}, pk_{ID})$ for the party ID and a non-interactive proof $pf_{ID}$ for $pk_{ID}$ (which may be required during key registration).

- $K \leftarrow \text{ShareKey}(ID_1, sk_{ID_1}, ID_2, pk_{ID_2})$: this algorithm takes as input an identity ID and a secret key $sk_{ID_1}$ along with another identity ID and corresponding public key $pk_{ID_2}$, and outputs either a shared key $K \in \mathcal{K}_{\text{NIKE}}$ for the two parties, or a failure symbol $\perp$. This algorithm is assumed to always output $\perp$ if input identities are not distinct.

For correctness, we require that, for an triple of identities $(ID_1, ID_2)$, and corresponding key pairs $(sk_{ID_1}, pk_{ID_1})$ and $(sk_{ID_2}, pk_{ID_3})$, algorithm ShareKey satisfies the constraint:

- $\text{ShareKey}(ID_1, sk_{ID_1}, ID_2, pk_{ID_2}) = \text{ShareKey}(ID_2, sk_{ID_2}, ID_1, pk_{ID_1})$

In this section we recall the CKS-light formal security model for a two party PKI-based non-interactive authenticated key-exchange (NIKE) protocol proposed in [FHKP13]. But we do slightly modification on modelling public key registration. Specifically, each party ID might be required to provide extra information (denoted by $pf_{ID}$) to prove the registered public key is sound. In practice, the concrete implementation of $pf$ is up to the CA [AFKM05] and may be either interactive or non-interactive. Examples can be found in RFC 4210 [AFKM05] and PKCS#10. Let $\{\text{Honest}, \text{Dishonest}\}$ be two vector lists.

**Security Experiment $\text{EXP}_{\text{NIKE,}\mathcal{A}}^{\text{cks-light}}(\kappa)$**: On input security parameter $1^\kappa$, the security experiment is proceeded as a game between a challenger $C$ and an adversary $\mathcal{A}$ based on a non-interactive key exchange protocol NIKE, where the following steps are performed:

1. The $C$ first run $pms^{\text{ni}}ke \leftarrow \text{NIKE.Setup}(1^\kappa)$ and gives $pms^{\text{ni}}ke$ to adversary $\mathcal{A}$.

2. The adversary $\mathcal{A}$ may interact with challenger $C$ with the following queries:
2 Preliminaries

- **RegisterHonest**\((\text{ID})\): On input an identity \(\text{ID} \in \mathcal{ID}\), if \(\text{ID} \notin \{\text{Honest, Dishonest}\}\) then \(C\) runs NIKE.KGen\((\text{pms}\text{\_nike}, \text{ID})\) to generate a long-term secret/public key pair \((\text{sk}_{\text{ID}}, \text{pk}_{\text{ID}}, \text{pf}_{\text{ID}}) \in (\mathcal{PK}, \mathcal{SK})\) and adds the tuple \((\text{ID}, \text{sk}_{\text{ID}}, \text{pk}_{\text{ID}}, \text{pf}_{\text{ID}})\) into the list \(\text{Honest}\), and returns \(\text{pk}_{\text{ID}}\) and \(\text{pf}_{\text{ID}}\) to \(A\); as otherwise a failure symbol \(\bot\) is returned. This query is allowed to query at most twice. Parties established by this query are called honest.

- **EstablishParty**\((\text{ID}, \text{pk}_{\text{ID}}, \text{pf}_{\text{ID}})\): This query allows the adversary to register an identity \(\text{ID}\) and a long-term public key \(\text{pk}_{\text{ID}}\) on behalf of a party \(\text{ID}\), if the \(\text{ID} \notin \{\text{Honest, Dishonest}\}\) and \(\text{pk}_{\text{ID}}\) is ensured to be sound by evaluating the non-interactive proof \(\text{pf}_{\text{ID}}\). We only require that the proof is non-interactive in order to keep the model simple. Parties established by this query are called dishonest, and the tuple \((\text{ID}, \text{pk}_{\text{ID}}, \text{pf}_{\text{ID}})\) would be added into the list \(\text{Dishonest}\).

- **RevealKey**\(\text{\_nike}(\text{ID}_1, \text{ID}_2)\): On input a tuple of registered identities \((\text{ID}_1, \text{ID}_2)\), \(C\) returns a failure symbol \(\bot\) if both parties \(\text{ID}_1\) and \(\text{ID}_2\) are dishonest. Otherwise \(C\) run ShareKey using the secret key of one of the honest parties in \((\text{ID}_1, \text{ID}_2)\) and the public key of the other party and returns the result to \(A\).

- **Test**\(\text{\_nike}(\text{ID}_1, \text{ID}_2)\): Given two identities \((\text{ID}_1, \text{ID}_2)\), the challenger \(C\) returns a failure symbol \(\bot\) if either \(\text{ID}_1 = \text{ID}_2\) or \(\text{ID}_1 \notin \text{Honest}\) or \(\text{ID}_2 \notin \text{Honest}\). Otherwise the challenger \(C\) samples a random bit \(b \leftarrow \{0, 1\}\), and it answers this query in terms of the bit \(b\). Specifically, if \(b = 1\), \(C\) runs ShareKey using the secret key of \(\text{ID}_1\) and the public key of \(\text{ID}_2\) to obtain the shared key \(K_1\); else if \(b = 0\), the challenger generates a random key \(K_1\). \(C\) returns \(K_1\) to adversary. This query can be asked only once.

3. Eventually, the adversary may terminate with outputting a bit \(b'\).

4. At the end of the experiment, 1 is returned if all following conditions hold:
   - \(A\) has issued a **Test**\(\text{\_nike}\) query without failure on input identities \((\text{ID}_1, \text{ID}_2)\),
   - Both parties \(\text{ID}_1\) and \(\text{ID}_2\) are honest and uncorrupted,
   - \(A\) has not issued **RevealKey** query on input identities \((\text{ID}_1, \text{ID}_2)\) in either order, and
   - \(b = b'\).
   Otherwise 0 is returned.

**Definition 8.** A two party non-interactive key exchange protocol \(\Sigma\) is called \((t, \epsilon_{\text{NIKE}})\)-secure if it holds that \(\Pr[\text{EXP}^{\text{\_nike}}_{\Sigma, A}(\kappa) = 1] - 1/2 \leq \epsilon_{\text{NIKE}}\) for all adversaries \(A\) running within time \(t\) in the above security experiment and for some negligible probability \(\epsilon_{\text{NIKE}} = \epsilon_{\text{NIKE}}(\kappa)\) in the security parameter \(\kappa\).
Note that if the query \texttt{EstablishParty}(ID\_\tau, pk\_ID\_\tau, pf\_ID\_\tau) is made with \( pf\_ID\_\tau = \emptyset \) then the above model equals the CKS-light model [FHKP13]; Otherwise it is slightly weaker than CKS-light model. The number of \texttt{EstablishParty} queries is bound by the time \( t \). In this model, we adopt a slightly variant CKS-light model for simplicity which is strong enough for our AKE construction.

## 2.7 Message Authentication Code

We first recall the notions of general message authentication codes scheme. A message authentication code \( \text{MAC} = (\text{MAC.KGen}, \text{MAC.Tag}) \) is a pair of algorithms with associated key space \( K_{\text{MAC}} \), message space \( M_{\text{MAC}} \) and tag space \( T_{\text{MAC}} \).

- \( K \leftarrow \text{MAC.KGen}(1^n) \): this is a probabilistic key-generation algorithm which takes as input a security parameter \( 1^n \) and outputs a secret key \( K \in K_{\text{MAC}} \).
- \( tg \leftarrow \text{MAC.Tag}(K, m) \): this is a deterministic algorithm which takes as input a secret key \( K \in K_{\text{MAC}} \) and a message \( m \in M_{\text{MAC}} \) and outputs an authentication tag \( tg \in T_{\text{MAC}} \).

We assume that verification of tags is done in the canonical way, i.e. given the tag \( tg \) and message \( m \) the verifier accepts the tag if \( tg = \text{MAC.Tag}(K, m) \).

**Definition 9.** We say that a message authentication code scheme \( \text{MAC} = (\text{MAC.KGen}, \text{MAC.Tag}) \) is \((q, t, \epsilon_{\text{MAC}})\)-secure against forgeries under adaptive chosen-message attacks, if probability bound \( \Pr[\text{EXP}_{\text{MAC}, A}(\kappa) = 1] \leq \epsilon_{\text{MAC}} \) holds for all adversaries \( A \) running in time at most \( t \) in the following experiment.

\[
\begin{align*}
K & \leftarrow \text{MAC.KGen}(1^n); \\
(m^*, tg^*) & \leftarrow A^{\text{MAC}(K, \cdot)}, \text{which can make up to } q \text{ queries to oracle } \text{MAC}(K, \cdot); \\
\text{return } 1, \text{if the following conditions are held:} \\
1. & \quad tg^* = \text{MAC.Tag}(K, m^*), \text{and} \\
2. & \quad A \text{ didn’t submit } m^* \text{ to } \text{MAC}(K, \cdot) \text{ oracle;} \\
\text{output } 0, \text{otherwise;}
\end{align*}
\]

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

If a MAC scheme is \((1, t, \epsilon_{\text{MAC}})\)-secure, then it is known as one-time message authentication code (OTMAC) scheme. Namely each authentication key of OTMAC scheme is used to authenticate a single message. In our AKE compiler shown in Section 4.3, we only require one-time message authentication code scheme. It is easy to see that notion
of one-time MAC is weaker than the notion of regular MAC that each authentication key can be used to authenticate more than one message. Concrete constructions of OTMAC scheme can be based, for instance, on pairwise independent hash functions [WC81].

2.8 Collision-Resistant Hash Functions

Let \( \text{CRHF} : K_{\text{CRHF}} \times M_{\text{CRHF}} \rightarrow Y_{\text{CRHF}} \) be a family of keyed-hash functions where \( K_{\text{CRHF}} \) is the key space, \( M_{\text{CRHF}} \) is the message space and \( Y_{\text{CRHF}} \) is the hash value space. The hash key \( h_{\text{CRHF}} \in K_{\text{CRHF}} \) defines a hash function, denoted by \( \text{CRHF}(h_{\text{CRHF}}, \cdot) \), which is generated by a PPT algorithm \( \text{CRHF.KGen}(1^\kappa) \) on input security parameter \( \kappa \). On input a message \( m \in M_{\text{CRHF}} \), this function \( \text{CRHF}(h_{\text{CRHF}}, m) \) generates a hash value \( y \in Y_{\text{CRHF}} \).

**Definition 10.** CRHF is called \((t, \epsilon_{\text{CRHF}})\)-secure if all \( t \)-time adversaries \( A \) have negligible advantage \( \epsilon_{\text{CRHF}} = \epsilon_{\text{CRHF}}(\kappa) \) with

\[
\Pr \left[ h_{\text{CRHF}} \leftarrow \text{CRHF.KGen}(1^\kappa), (m, m') \leftarrow A(1^\kappa, h_{\text{CRHF}}); \right.
\]
\[
m \neq m', (m, m') \in M_{\text{CRHF}},
\]
\[
\text{CRHF}(h_{\text{CRHF}}, m) = \text{CRHF}(h_{\text{CRHF}}, m') \right] \leq \epsilon_{\text{CRHF}}
\]

where the probability is over the random coins of the adversary and \( \text{CRHF.KGen} \).

If the hash key \( h_{\text{CRHF}} \) is obvious from the context, we write \( \text{CRHF}(m) \) for \( \text{CRHF}(h_{\text{CRHF}}, m) \). In the construction of our AKE compiler in Section 4.3, we use CRHFs for the purpose of domain extension.

2.9 Target Collision-Resistant Hash Functions

Let \( \text{TCRHF} : K_{\text{TCRHF}} \times M_{\text{TCRHF}} \rightarrow Y_{\text{TCRHF}} \) be a family of keyed-hash functions associated with key space \( K_{\text{TCRHF}} \), message space \( M_{\text{TCRHF}} \) and hash value space \( Y_{\text{TCRHF}} \). The hash key \( h_{\text{TCRHF}} \in K_{\text{TCRHF}} \) of a hash function \( \text{TCRHF}(h_{\text{TCRHF}}, \cdot) \) is generated by a PPT algorithm \( \text{TCRHF.KGen}(1^\kappa) \) on input security parameter \( \kappa \). In the following, we recall the security notion [BR97, CS03] of Target Collision-Resistant Hash Functions.

**Definition 11.** TCRHF is called \((t, \epsilon_{\text{TCRHF}})\)-target-collision-resistant if for all \( t \)-time adversaries \( A \) it holds that

\[
\Pr \left[ h_{\text{TCRHF}} \leftarrow \text{TCRHF.KGen}(1^\kappa), m \leftarrow M_{\text{TCRHF}}, m' \leftarrow A(1^\kappa, h_{\text{TCRHF}}, m); \right.
\]
\[
m \neq m', m, m' \in M_{\text{TCRHF}},
\]
\[
\text{TCRHF}(h_{\text{TCRHF}}, m) = \text{TCRHF}(h_{\text{TCRHF}}, m') \right] \leq \epsilon_{\text{TCRHF}}
\]

where the probability is over the random bits of \( A \).
If the hash key $hk_{TCRHF}$ is obvious from the context, we write $TCRHF(m)$ for $TCRHF(hk_{TCRHF}, m)$. Note that the notion of target collision resistance is weaker than the notion of collision resistance. Our notion of TCRHF is also related to the stronger notion of universal one-way hash functions (UOWHF) [NY89]. But in the security experiment of the UOWHF, the target message $m$ is chosen by the adversary before having the hash key $hk_{TCRHF}$. As suggested in [CS03], normally target collision resistant functions can be realized with a specific cryptographic hash function such as MD5 and SHA. In our one-round AKE solutions, the TCRHF is enough to meet our construction requirement rather than regular CRHF.

2.10 Pseudo-Random Functions

The concept of pseudo-random functions is introduced by Goldreich, Goldwasser and Micali in [GGM84]. Let $PRF : \mathcal{K}_{PRF} \times \mathcal{D}_{PRF} \rightarrow \mathcal{R}_{PRF}$ denote a family of deterministic functions, where $\mathcal{K}_{PRF}$ is the key space, $\mathcal{D}_{PRF}$ is the domain and $\mathcal{R}_{PRF}$ is the range of $PRF$ for security parameter $\kappa$. Let $RL = \{(x_1, y_1), \ldots, (x_q, y_q)\}$ be a list which is used to record bit strings formed as tuple $(x_i, y_i) \in (\mathcal{D}_{PRF}, \mathcal{R}_{PRF})$ where $1 \leq i \leq q$ and $q \in \mathbb{N}$. On input a message $x \in \mathcal{D}_{PRF}$, the function $RF(x)$ is evaluated as follows:

- If $x \in RL$, then return corresponding $y \in RL$,
- Otherwise return $y \leftarrow \mathcal{R}_{PRF}$ and record $(x, y)$ into $RL$.

Definition 12. We say that $PRF$ is a $(q, t, \epsilon_{PRF})$-secure pseudo-random function family, if it holds that $|\Pr[EXP_{ind-cma,PRF,\mathcal{A}}^{ind-cma}(\kappa) = 1] - 1/2| \leq \epsilon_{PRF}$ for all adversaries $\mathcal{A}$ that make a polynomial number of oracle queries $q$ while running in time at most $t$ in the following experiment:

\[
\begin{align*}
\mathcal{F}(b, x) &\quad \text{If } x \notin \mathcal{D}_{PRF} \text{ then return } \bot; \\
b \leftarrow \{0, 1\}, k \leftarrow \mathcal{K}_{PRF}; &\quad \text{If } b = 1 \text{ then return } PRF(k, x); \\
b' \leftarrow \mathcal{A}^{\mathcal{F}(b, \cdot)}(\kappa); &\quad \text{Otherwise return } RF(x); \\
\text{If } b = b' \text{ then return } 1; &\quad \text{Otherwise return } 0;
\end{align*}
\]

where $\epsilon_{PRF} = \epsilon_{PRF}(\kappa)$ is a negligible function in the security parameter $\kappa$, and the number of allowed queries $q$ is bound by $t$.

2.11 Min-entropy and Strong Randomness Extractors

Definition 13. We say that a random variable $X^*$ distributed over domain $\mathcal{M}$ has min-entropy $\kappa$, if for all $X \in \mathcal{M}$ it holds that $\Pr[X^* = X] \leq 2^{-\kappa}$.
The min-entropy is a formal measurement for 'good' key distribution of a key (see also [GKR04, Definition 2]), for example the key of following KE scheme, KEM scheme and NIKE scheme.

Let $SEXT : S_{SEXT} \times M_{SEXT} \rightarrow R_{SEXT}$ be a function family associated with seed space $S_{SEXT}$, domain $M_{SEXT}$, and range $R_{SEXT}$. We recall the formal definition for strong randomness extractor following literature [NZ96].

**Definition 14.** We say that a function family $SEXT$ is a $(\kappa, \epsilon_{SEXT})$-strong randomness extractor, if for any variable $X$ distributed over $M_{SEXT}$ that has min-entropy $\kappa$ and for any seed $k_{SEXT}$ which is chosen uniformly at random from $S_{SEXT}$ and for any value $R$ which is chosen uniformly at random from $R_{SEXT}$, the two distributions $\langle k_{SEXT}, SEXT(k_{SEXT}, X) \rangle$ and $\langle k_{SEXT}, R \rangle$ have statistical distance at most $\epsilon_{SEXT}$, i.e.

$$\epsilon_{SEXT} = \max_{T \subseteq R_{SEXT}} \left( \Pr[SEXT(k_{SEXT}, X) \in T] - \Pr[R \in T] \right)$$

$$= \frac{1}{2} \sum_{y \in R_{SEXT}} \left| \Pr[SEXT(k_{SEXT}, X) = y] - \Pr[R = y] \right| .$$

In the context where the seed $k_{SEXT}$ is clear we will write $SEXT(X)$ for $SEXT(k_{SEXT}, X)$.

The Definition 14 also implies that $\epsilon_{SEXT} = \left| \Pr[D(SEXT(X)) = 1] - \Pr[D(R) = 1] \right|$ for any PPT distinguishers $D$ and $\Pr[D(SEXT(X)) = 1], \Pr[D(R) = 1]$ are the probabilities that $D$ outputs 1 when given a sample of $SEXT(X)$ over message $X \in M_{SEXT}$ and random variable $R \in R_{SEXT}$, respectively, these probabilities being computed over the randomness in $X$ and $R$ and over $D$’s internal coins.

As suggested by Dodis et al. [DGH+04], one could use a pseudo-random function as a strong randomness extractor. Some good results on key derivation and randomness extraction can be also found in [CFG05].

### 2.12 Tag-based Authentication Schemes

In this section we propose an extended definition of *tag-based authentication schemes* (TBAS). Our definition encompasses a wider range of authentication schemes, in particular those where the verifier maintains some secret state information in the entire course of the protocol.

We will for simplicity focus on two-round (challenge and response based) authentication protocols. A tag-based authentication scheme is a triple of PPTs: $TBAS = (TAG, TAP, TAV)$, where $TAG$ is a probabilistic set-up procedure generating the public and secret key material of a prover, $TAP$ is (a possibly probabilistic) algorithm that is run by the prover. $TAV = (TAV.ch, TAV.vfy)$ consist of two algorithms that are executed by the verifier.

Firstly, each prover $ID_p$ may run $(sk_{ID_p}, pk_{ID_p}) \xleftarrow{} TAG(1^\kappa)$ to generate a pair of private/public keys $sk_{ID_p}$ and $pk_{ID_p}$ (which could be either long-term or ephemeral).
Each verifier $ID_v$ may run $(sk_{ID_v}, pk_{ID_v}) \rightarrow \mathcal{TAG}(1^\kappa)$ to generate a pair of private/public keys $sk_{ID_v}$ and $pk_{ID_v}$. As usual $(pk_{ID_v}, pk_{ID_v})$ are made public. During the protocol execution phase, prover $ID_p$ and verifier $ID_v$ execute the protocol as follows:

1. $ID_v$ first runs $TAV.ch(pk_{ID_p}, t)$ to compute a challenge $c$ and a corresponding secret state $st$ as $(c, st) = TAV.ch(pk_{ID_p}, t)$ from the public key of $pk_{ID_p}$ and a tag $t$. The challenge $c$ is sent to the prover $ID_p$. $st$ is kept secret. We note that for some protocols $st$ can possibly be empty.

2. The prover $ID_p$ computes a response $r := TAP(sk_{ID_p}, c, t)$ for tag $t$, and sends the $r$ to $ID_v$.

3. The verifier $ID_v$ verifies the response by running $TAV.vfy(pk_{ID_p}, st, t, c, r)$, which outputs accept, if and only if $r$ is a valid response to $c$ and tag $t$; otherwise reject is returned.

We consider the security of the TA protocol by running an experiment with an adversary $A$ as follows.

**EXP_{TA,A}(\kappa)**

- In the security experiment, $A$ is allowed to access to oracles $O_P$, $O_{ch}$ and $O_{vfy}$, implementing the algorithms $TAP$ and $TAV = (TAV.ch, TAV.vfy)$ of the protocol.

- The adversary receives as input the public key $pk_{ID_p}, pk_{ID_v}$ of a prover/receiver generated via $(sk_{ID_p}, pk_{ID_v}) \rightarrow \mathcal{TAG}(1^\kappa)$ and (if necessary) $(sk_{ID_v}, pk_{ID_v}) \rightarrow \mathcal{TAG}(1^\kappa)$. She may submit at most $q$ tags $(t_1, t_2, \ldots, t_q)$ to the verifier’s oracle $O_{ch}$, which returns $(c_i, st_i) := TAV.ch(pk_{ID_p}, t_i)$ for $i \in 1, 2, \ldots, q$ to adversary.

- Meanwhile $A$ may also submit at most $q$ challenge-tag pairs $(c'_1, t'_1), (c'_2, t'_2), \ldots, (c'_q, t'_q)$ to the prover’s oracle $O_P$, which responds with respective authentication token $r'_i \rightarrow TAP(sk_{ID_p}, c'_i, t'_i)$ for $i \in 1, 2, \ldots, q$.

- Additionally $A$ may submit at most $q$ tuples $(st''_i, t''_i, c''_i, r''_i)$ to the verification oracle $O_{vfy}$, which responds with accept if $TAV.vfy(pk_{ID_p}, st''_i, t''_i, c''_i, r''_i)$ outputs accept and reject otherwise.

- At some point, $A$ may query $O_{ch}$ by submitting a tag $t^*$ and a special symbol $\top$, then the oracle responds with a new challenge $c^*$ that is computed via $(c^*, st^*) := TAV.ch(pk_{ID_p}, t^*)$ but keeps the $st^*$ as secret.

- Eventually, $A$ may terminate and return a response $r^*$.

- Finally, 1 is returned if accept $= O_{vfy}(pk_{ID_p}, st^*, t^*, c^*, r^*)$ and $t^* \neq \{t_1, \ldots, t_q\}$; otherwise 0 is returned.
Definition 15. We say that a (two-round) tag-based authentication protocol TA is $(q,t,\epsilon_{TA})$-secure, if it holds that $|\Pr[\text{EXP}_{TA,A}(\kappa) = 1]| \leq \epsilon_{TA}$ for all adversary $A$ running in probabilistic polynomial time $t$ where $\epsilon_{TA} = \epsilon_{TA}(\kappa)$ is a negligible function in the security parameter $\kappa$.

2.13 Weak Programmable Hash Functions

We review the notions of weak programmable hash functions introduced by Hofheinz et al. [HJK11] (which is specific form of Programmable Hash Functions [HK08]). Let $G_\kappa$ be a family of cyclic groups associated with security parameter $\kappa$, where the subscript might be omitted for simplicity if $\kappa$ is clear in the context. We assume a group hash function $\text{PHF} = (\text{PHF.Gen, PHF.Eval})$ over $G$ consists of two efficient algorithms with following semantics:

- $hk_{\text{PHF}} \xleftarrow{\$} \text{PHF.Gen}(1^\kappa)$: on input the security parameter $1^\kappa$, the key generation algorithm generates a hash function key $hk_{\text{PHF}} \in K_{\text{PHF}}$ from space $K_{\text{PHF}}$.

- $y \leftarrow \text{PHF.Eval}(hk_{\text{PHF}}, X)$: the deterministic evaluation algorithm takes as input the hash key $hk_{\text{PHF}}$ and a message $X \in M_{\text{PHF}}$, and returns hash value $y \in G$, where $M_{\text{PHF}}$ is a message space.

All spaces are parameterized with security parameter $\kappa$.

Definition 16. A group hash function $\text{PHF} = (\text{PHF.Gen, PHF.Eval})$ is a weak $(m,n,\psi,\rho)$-programmable hash function, if there is an efficient probabilistic algorithm $\text{PHF.TrapGen}$ and an efficient deterministic algorithm $\text{PHF.TrapEval}$ with following properties:

1. $(hk_{\text{PHF}}, tk_{\text{PHF}}) \xleftarrow{\$} \text{PHF.TrapGen}(1^\kappa, g, u, X_1, \ldots, X_m)$: The probabilistic trapdoor generation algorithm takes as input the security parameter $1^\kappa$, group generators $g, u \in G$ and $X_1, \ldots, X_m \in M_{\text{PHF}}$, and generates a hash function key $hk_{\text{PHF}}$ together with trapdoor $tk_{\text{PHF}}$.

2. $\forall (g, u) \in G$, the hash function keys $hk_{\text{PHF}} \xleftarrow{\$} \text{PHF.Gen}(1^\kappa)$ and $hk'_{\text{PHF}} \xleftarrow{\$} \text{PHF.TrapGen}(1^\kappa, g, u, X_1, \ldots, X_m)$ are statistically $\psi$-close.

3. $(a_X, b_X) \leftarrow \text{PHF.TrapEval}(tk_{\text{PHF}}, X)$: On input trapdoor $tk_{\text{PHF}}$ and $X \in M_{\text{PHF}}$, the deterministic trapdoor evaluation algorithm outputs integers $a_X, b_X$ such that for any $X$,
$$\text{PHF.Eval}(hk_{\text{PHF}}, X) = u^{a_X} g^{b_X}.$$

4. $\forall (g, h) \in G$, $\forall hk_{\text{PHF}} \xleftarrow{\$} \text{PHF.TrapGen}(1^\kappa, g, u)$, and $\forall (X_1, \ldots, X_m, Z_1, \ldots, Z_n) \in M_{\text{PHF}}$ such that $X_i \neq Z_j$ for any $i, j$, we have that
$$\Pr[a_{X_1} = \cdots = a_{X_m} = 0 \text{ and } a_{Z_1}, \ldots, a_{Z_n} \neq 0] \geq \rho.$$
where \((a_{X_i}, b_{X_i}) \leftarrow \text{PHF.TrapEval}(t_{\text{PHF}}, X_i)\) and \((a_{Z_j}, b_{Z_j}) \leftarrow \text{PHF.TrapEval}(t_{\text{PHF}}, Z_j)\), and the probability is over the random coins of the adversary and \(\text{PHF.TrapGen}\).

We say that \(\text{PHF}\) is a weak \((m,n)\)-programmable hash function for short if \(\psi\) is negligible and \(\rho\) is noticeable. In addition, we say that \(\text{PHF}\) is \((m,ploy)\)-programmable, if \(\text{PHF}\) is \((m,q)\)-programmable for any polynomial \(q = q(k)\). In the security proof simulation, the group generator \(u\) is normally used to embed challenge value from the challenge instance of certain hard problem.

### 2.14 Bilinear Groups

In the following, we briefly recall some of the basic properties of bilinear groups. Our AKE solutions mainly consist of elements from a single group \(G\). We therefore concentrate on symmetric bilinear maps. The bilinear groups will be parameterized by a symmetric pairing parameter generator, denoted by \(\text{PG.Gen}\). This is a polynomial time algorithm that on input a security parameter \(1^{\kappa}\), returns the description of two multiplicative cyclic groups \(G\) and \(G_T\) of the same prime order \(p\), generator \(g\) for \(G\), and a bilinear computable pairing \(e : G \times G \rightarrow G_T\).

**Definition 17** (Symmetric Bilinear groups). We call \(\mathcal{P}G = (G, g, G_T, p, e)\) be a set of symmetric bilinear groups, if the function \(e\) is an (admissible) bilinear map and it holds that:

1. **Bilinear:** \(\forall (a, b) \in G\) and \(\forall (x, y) \in \mathbb{Z}_p\), we have \(e(a^x, b^y) = e(a, b)^{xy}\).
2. **Non-degenerate:** \(e(g, g) \neq 1_{G_T}\), is a generator of group \(G_T\).
3. **Efficiency:** \(\forall (a, b) \in G\), \(e\) is efficiently computable.

### 2.15 Multilinear Groups

In the following, we recall the definition of symmetric multilinear groups introduced in [BS02]. We assume that a party can call a group generator \(\text{MLG.Gen}(1^{\kappa}, n)\) to obtain a set of multilinear groups. On input a security parameter \(\kappa\) and a positive integer \(2 < n \in \mathbb{N}\), the polynomial time group generator \(\text{MLG.Gen}(1^{\kappa}, n)\) returns two multiplicative cyclic groups \(G\) and \(G_T\) of the same prime order \(p\), generator \(g\) for \(G\), and a \(n\)-multilinear map \(me : G^n \times G \rightarrow G_T\).

We summarize the properties of \(n\)-multilinear groups in the following definition.

**Definition 18** (Multilinear groups). We call \(\mathcal{M}LG = (G, G_T, p, me)\) be symmetric multilinear groups, if the \(n\)-multilinear map \(me\) holds that:
2 Preliminaries

1. **n-multilinear**: \( \forall (c_1, \ldots, c_n) \in G_1 \) and \( \forall (y_1, \ldots, y_n) \in \mathbb{Z}_p \), we have 
   \[ me(c_1^{y_1}, \ldots, c_n^{y_n}) = me(c_1, \ldots, c_n)^{y_1 \cdots y_n}. \]

2. **Non-degenerate**: \( me(g, \ldots, g) \neq 1_{G_1} \), is a generator of group \( G_1 \).

3. **Efficiency**: \( \forall (c_1, \ldots, c_n) \in G_1 \), the operation \( me(c_1, \ldots, c_n) \) is efficiently computable.

Most recently, Garg, Gentry, and Halvey [GGH13, GGH12] announced a surprising result on multilinear maps at Eurocrypt2013 conference: Exploiting ideal lattices they produced a candidate mechanism that would approximate multilinear maps for many applications. We here just focus on a general definition of symmetric n-multilinear groups without loss of generality.

## 2.16 Complexity Assumptions

### 2.16.1 Bilinear Decisional Diffie-Hellman Assumption

Let \( PG = (G, g, G_T, p, e) \xleftarrow{} PG.Gen(1^\kappa) \) denote the description of symmetric bilinear groups as Definition 17. The Bilinear Decisional Diffie-Hellman (BDDH) problem has originally been introduced by Joux [Jou00] and later formalized by Boneh and Franklin [BF03]. We first consider the following traditional version of Bilinear Decisional Diffie-Hellman (BDDH) problem for symmetric pairing. The BDDH problem is stated as follows: given tuple \((g, g^a, g^b, g^c, e(g, g)^\gamma)\) for \((a, b, c, \gamma) \in (\mathbb{Z}_p^*)^4\) as input, output 1 if 
\[ e(g, g)^\gamma = e(g, g)^{abc} \] and 0 otherwise.

**Definition 19.** We say that the BDDH problem relative to generator \( PG.Gen(1^\kappa) \) is \((t, \epsilon_{\text{BDDH}})\)-hard, if the probability bound 
\[ |Pr[\text{EXP}_{PG.Gen,A}(\kappa) = 1] - 1/2| \leq \epsilon_{\text{BDDH}} \] holds for all adversaries \( A \) running in probabilistic polynomial time \( t \) in the following experiment:

\[ \text{EXP}_{PG.Gen,A}(\kappa) \]
\[ PG = (G, g, G_T, p, e) \xleftarrow{} PG.Gen(1^\kappa); \]
\[ (a, b, c, \gamma) \xleftarrow{} \mathbb{Z}_p^*; \]
\[ b \xleftarrow{} \{0, 1\}, \text{ if } b = 1 \Gamma \leftarrow e(g, g)^{abc}, \text{ otherwise } \Gamma \leftarrow e(g, g)^\gamma; \]
\[ b' \leftarrow A(1^\kappa, PG, g^a, g^b, g^c, \Gamma); \]
\[ \text{if } b = b' \text{ then return } 1, \text{ otherwise return } 0; \]

where \( \epsilon_{\text{BDDH}} = \epsilon_{\text{BDDH}}(\kappa) \) is a negligible function in the security parameter \( \kappa \).

### 2.16.2 Cube Bilinear Decisional Diffie-Hellman Assumption

Let \( PG = (G, g, G_T, p, e) \xleftarrow{} PG.Gen(1^\kappa) \) denote the description of symmetric bilinear group as Definition 17. The Cube Bilinear Decisional Diffie-Hellman (CBDDH) problem
that is stated as follows: given tuple \((g, g^a, e(g, g)\gamma)\) for \((a, \gamma) \in (\mathbb{Z}_p^*)^2\) as input, output 1 if \(e(g, g)\gamma = e(g, g)^a\) and 0 otherwise.

**Definition 20.** We say that the \(\text{CBDDH}\) problem relative to generator \(\text{PG.Gen}\) is \((t, \epsilon_{\text{CBDDH}})\)-hard, if the probability bound \(|\Pr[\text{EXP}_{\text{PG.Gen}}^{\text{cbddh}}(\kappa, n) = 1] - 1/2| \leq \epsilon_{\text{CBDDH}}\) holds for all adversaries \(A\) running in probabilistic polynomial time \(t\) in the following experiment:

\[
\begin{align*}
\text{EXP}_{\text{PG.Gen}}^{\text{cbddh}}(\kappa, n) & \\
\text{PG} &= (G, G_T, g, p, e) \xleftarrow{\$} \text{PG.Gen}(1^\kappa); \\
(a, \gamma) & \xleftarrow{\$} \mathbb{Z}_p^*; \\
b & \xleftarrow{\$} \{0, 1\}, \text{ if } b = 1 \Gamma \leftarrow e(g, g)^a, \text{ otherwise } \Gamma \leftarrow e(g, g)^\gamma; \\
b' & \leftarrow A(1^\kappa, \text{PG}, g^a, \Gamma); \\
\text{if } b = b' \text{ then return } 1, \text{ otherwise return } 0;
\end{align*}
\]

where \(\epsilon_{\text{CBDDH}} = \epsilon_{\text{CBDDH}}(\kappa)\) is a negligible function in the security parameter \(\kappa\).

### 2.16.3 n-Multilinear Decisional Diffie-Hellman Assumption

We present a generalization of the \(\text{CBDDH}\) assumption in \(n\)-multilinear groups \(\mathcal{MLG} = (G, G_T, g, p, me) \xleftarrow{\$} \text{MLG.Gen}(1^\kappa, n)\) that we call the \(n\)-Multilinear Decisional Diffie-Hellman (nMDDH) assumption. Roughly speaking, the nMDDH problem is stated as follows: given tuple \((g, g^a, me(g, \ldots, g)^\gamma)\) for \((a, \gamma) \in (\mathbb{Z}_p^*)^2\) as input, output 1 if \(me(g, \ldots, g)^\gamma = me(g, \ldots, g)^{a_{n+1}}\) and 0 otherwise.

**Definition 21.** We say that the nMDDH problem relative to generator \(\text{MLG.Gen}\) is \((t, \epsilon_{\text{nMDDH}})\)-hard, if the probability bound \(|\Pr[\text{EXP}_{\text{PG.Gen}}^{\text{nmddh}}(\kappa, n) = 1] - 1/2| \leq \epsilon_{\text{nMDDH}}\) holds for all adversaries \(A\) running in probabilistic polynomial time \(t\) in the following experiment:

\[
\begin{align*}
\text{EXP}_{\text{PG.Gen}}^{\text{nmddh}}(\kappa, n) & \\
\mathcal{MLG} &= (G, G_T, g, p, me) \xleftarrow{\$} \text{MLG.Gen}(\kappa, n); \\
(a, \gamma) & \xleftarrow{\$} \mathbb{Z}_p^*; \\
b & \xleftarrow{\$} \{0, 1\}, \text{ if } b = 1 \Gamma \leftarrow me(g_1, \ldots, g_1)^{a_{n+1}}, \text{ otherwise } \Gamma \leftarrow me(g, \ldots, g)^\gamma; \\
b' & \leftarrow A(1^\kappa, \mathcal{MLG}, g^a, \Gamma); \\
\text{if } b = b' \text{ then return } 1, \text{ otherwise return } 0;
\end{align*}
\]

where \(\epsilon_{\text{nMDDH}} = \epsilon_{\text{nMDDH}}(\kappa)\) is a negligible function in the security parameter \(\kappa\).
3 Security Models for Authenticated Key Exchange

In this section we present formal security models for authenticated key exchange (AKE) protocols that cover not only two party AKE protocols but also group AKE protocols. In these models, while emulating the real-world capabilities of an active adversary, we provide an ‘execution environment’ for adversaries following an important line of research [CK01, KY03, LLM07, MSU09, JKSS12] which is initiated by Bellare and Rogaway [BR94]. The capabilities of an adversary are formulated in a strong sense who is provided with enormous power to take full control over the communication network, e.g. altering or injecting messages as she wishes. In particular she may compromise long-term keys of parties or secret states of protocol instances at any time. In the sequel we will adopt unified execution environment and adversarial model for these models, which are extended from the framework of [JKSS12].

3.1 Execution Environment

In the execution environment, we fix a set of honest parties \( \{ID_1, \ldots, ID_\ell \} \) for \( \ell \in \mathbb{N} \), where \( ID_i \in IDS \) is the identity of a party. Each identity \( ID_i \) is associated with a long-term key pair \( (sk_{ID_i}, pk_{ID_i}) \in (SK, PK) \). Note that those identities are also indexed via variable \( i \in [\ell] \). For identity and public key registration, each party \( ID_i \) might be required to provide extra information (denoted by a variable \( pf \)) to prove that the public key is sound (for instance the adversary is ensured to either know the secret key or registered public key is correctly formed). In practice, the concrete specification of \( pf \) is up to CA [AFKM05] and may be either interactive or non-interactive. Examples can be found in RFC 4210 [AFKM05] and PKCS#10 [RDS00]. In this model we only focus on non-interactive proof for simplicity. Each honest party \( ID_i \) can sequentially and concurrently execute the protocol multiple times with different intended partners, this is characterized by a collection of oracles \( \{\pi^s_{ID_i} : i \in [\ell], s \in [\rho]\} \) for \( \rho \in \mathbb{N} \).\(^1\) Oracle \( \pi^s_{ID_i} \) behaves as party \( ID_i \) carrying out a process to execute the \( s \)-th session (protocol instance), which has access to the long-term key pair \( (sk_{ID_i}, pk_{ID_i}) \) of party \( ID_i \) and to all public keys of other parties. Moreover, we assume each oracle \( \pi^s_{ID_i} \) maintains a list of independent internal state variables with following semantics:

\(^1\)An oracle in this thesis might be alternatively written as \( \pi^s_{ID_i} \) which is conceptually equivalent to \( \pi^s_i \).
• \( \text{pid}_i^s \) – A variable stores a set of partner identities in the group with whom \( \pi_i^s \) intends to establish a session key (including \( \text{ID}_i \) itself), where the identities are ordered lexicographically.

• \( \Phi_i^s \) – A variable stores the oracle decision \( \Phi_i^s \in \{\text{accept, reject}\} \).

• \( K_i^s \) – A variable records the session key \( K_i^s \in K_{\text{AKE}} \).

• \( st_i^s \) – A variable stores the maximum secret session states that are allowed to be leaked.

• \( \text{sid}_i^s \) – A variable stores the session identifier which is used to uniquely identify a protocol instance at party \( \text{ID}_i \) and is assumed to be derived during the run of \( \pi_i^s \).

All those variables of each oracle are initialized with empty string denoted by symbol \( \emptyset \). At some point, each oracle \( \pi_i^s \) may complete the execution always with a decision state \( \Phi_i^s \in \{\text{accept, reject}\} \). We assume that the session key is assigned to the variable \( K_i^s \) (such that \( K_i^s \neq \emptyset \)) iff oracle \( \pi_i^s \) has reached an internal state \( \Phi_i^s = \text{accept} \). In addition, we also assume that the internal states \( \text{sid}_i^s \neq \emptyset \) and \( \text{pid}_i^s \neq \emptyset \) when \( \Phi_i^s = \text{accept} \).

To formalize the notion that two oracles are engaged in an on-line communication, we define the partnership via a notion of matching sessions.

**Definition 22** (Matching sessions). We say that an oracle \( \pi_i^s \) has a matching session to another oracle \( \pi_j^t \), if both oracles satisfy that \( \text{pid}_i^s = \text{pid}_j^t \) and \( \text{sid}_i^s = \text{sid}_j^t \).

**Definition 23** (Correctness). Let \( \pi_i^s \) and \( \pi_j^t \) be two oracles. We say an authenticated key exchange protocol \( \Sigma \) is correct, if both oracles \( \pi_i^s \) and \( \pi_j^t \) accept such that \( \pi_i^s \) and \( \pi_j^t \) have matching sessions, then it holds that \( K_i^s = K_j^t \).

The above correctness definition is applicable to either two party AKE protocol or group AKE protocol.

### 3.2 Adversarial Model

An adversary \( \mathcal{A} \) in our model is a PPT Turing Machine taking as input the security parameter \( 1^k \) and the public information (e.g. generic description of above environment), which may interact with these oracles by issuing the following queries.

• **Send**(\( \pi_i^s, m \)): The adversary can use this query to send any message \( m \) of his own choice to oracle \( \pi_i^s \). The oracle will respond with the next message \( m^* \) (if any) to be sent according to the protocol specification. Oracle \( \pi_i^s \) would be initiated via sending the oracle the first message \( m = (\top, \text{pid}_i^s) \) consisting of a special initialization symbol \( \top \) and a variable storing identities of participants.\(^2\) After

\(^2\)If the protocol can work under post-specified peer setting [CK02, MU08], then the \( \text{pid}_i^s \) in the first message could be empty string \( \emptyset \).
answering a Send query, the variables \((\text{pid}^s_i, \Phi^s_i, K^s_i, st^s_i, \text{sid}^s_i)\) might be updated depending on the specific protocol.

- **RevealKey**\((\pi^s_i)\): Oracle \(\pi^s_i\) responds with the contents of variable \(K^s_i\).
- **StateReveal**\((\pi^s_i)\): Oracle \(\pi^s_i\) responds with the contents of variable \(st^s_i\).
- **Corrupt**\((\text{ID}_i)\): Oracle \(\pi^1_i\) responds with the long-term secret key \(sk_{\text{ID}_i}\) of party \(\text{ID}_i\) if \(i \in [\ell]\) otherwise a failure symbol ⊥ is returned.
- **EstablishParty**\((\text{ID}_\tau, pk_{\text{ID}_\tau}, pf_{\text{ID}_\tau})\): This query allows the adversary to register an identity \(\text{ID}_\tau\) \((\ell < \tau < N)\) and a static public key \(pk_{\text{ID}_\tau}\), if \(\text{ID}_\tau\) is unique and \(pk_{\text{ID}_\tau}\) is ensured to be sound by evaluating the non-interactive proof \(pf_{\text{ID}_\tau}\). We only require that the proof is non-interactive in order to keep the model simple. Parties established by this query are called dishonest.
- **Test**\((\pi^s_i)\): This query may only be asked once throughout the experiment. Oracle \(\pi^s_i\) handles this query as follows: If the oracle has state \(\Phi^s_i = \text{reject}\) or \(K^s_i = \emptyset\), then it returns some failure symbol ⊥. Otherwise it flips a fair coin \(b\), samples a random element \(K_0\) from key space \(\mathcal{K}_{\text{AKE}}\), sets \(K_1 = K^s_i\) to the real session key, and returns \(K_b\).

The EstablishParty query is widely used to model the chosen identity and public key attacks in different security models. In this query, the detail form of \(pf\) (i.e. regarding how to register an identity and corresponding public key) should be specified by each protocol (which may corresponds to the proof of knowledge assumptions for public key registration as discussed in [LOS+06, BN06]). Please note that if the protocol allows for arbitrary key registration then one could set the parameter \(pf = \emptyset\). For simplicity, the Corrupt query only models the compromise of the long-term private key of a honest party, e.g. by a malware attack or by a side-channel attacks on a secure device where the long-term private keys are stored. We stress that, via issuing Corrupt query, an adversary is only able to learn long-term private keys of parties which are honest, i.e. the parties which are not established by adversary EstablishParty query. This follows the fact that those dishonest parties are controlled by adversary herself. Besides eCK models in literatures, similar modelling approach concerning EstablishParty query and Corrupt query can be also found in [FHKP13].

**Implementation Model vs. Session States.** In order to fulfil the gap that often exists between formal models and practical security, Sarr et al. [SEVB10] introduced two implementation scenarios for the situation that at each party an untrusted host machine is used together with a secure device such as smart card. The modelling techniques involving secure device were also previously used by Bresson et al. [BCP02]. A secure device may usually be used to store long-term cryptographic authentication keys and
at least be able to accomplish a library of mathematical functions (such as addition, modulo and exponentiation) which are necessary to implement cryptographic operations or primitives. Hence based on secure device we are able to adopt a ‘All-and-Nothing’ strategy to define the states that can be revealed without leaving any ambiguity. General speaking we could assume that all intermediate states and ephemeral keys generated on host machine are susceptible to StateReveal query to model the maximum state leakage (MSL) attacks, but we treat the secure device as a black-box which is immune to leakage of internal states.\textsuperscript{3} Of course one could distribute all protocol computations on the secure device then the security model would equal to a model without StateReveal query. However the security result of a protocol analysed with such implementation scenario must be weaker than that in a case allowing leakage of states. In contrast, our goal is to define the maximum states that can be leaked. As those secure devices might be short in both storage capacity and computational resource, the algorithm on secure device is often causing performance bottleneck of systems. In addition, the communication round between host machine and secure device (which may be called HS-round for short) might cause another efficiency problem, since the serial I/O bus of most secure devices is too slow. Those facts make it necessary for us to optimize AKE protocols when they are realized involving secure device.

We stress that the exact meaning of the StateReveal must be defined by each protocol separately in conjunction of certain implementation scenario involving secure device, and each protocol should be proven secure to resist with such kind of state leakage as claimed. Namely a protocol should specify the content stored in the variable $st^s_i$ during protocol execution. In other word, each protocol should define the protocol steps processed on secure device in terms of above ‘All-and-Nothing’ strategy. To make our model simple, we only use a unified StateReveal query rather than using different kinds of such queries with various ‘aliasases’. Since no matter how many different kinds of ‘StateReveal’ are modeled, each protocol should specify which ‘StateReveal’ query it can resist with in terms of specific implementation scenario.

AKE SECURITY GAME. The security game is played between a challenger $C$ and an adversary $A$, where the following steps are performed:

1. At the beginning of the game, the challenger $C$ implements the collection of oracles $\{\pi^s_i : i \in [\ell], s \in [\rho]\}$, and generates $\ell$ long-term key pairs $(pk_{ID_i}, sk_{ID_i})$ and corresponding proof $pf_i$ for all honest parties $ID_i$ for $i \in [\ell]$ where the identity $ID_i \in ID_S$ of each party is chosen uniquely. $C$ gives adversary $A$ all identities, public keys and corresponding proofs $\{(ID_1, pk_{ID_1}, pf_{ID_1}), \ldots, (ID_\ell, pk_{ID_\ell}, pf_{ID_\ell})\}$ as input.

\textsuperscript{3}Although there might exist some side-channel attacks (such as [BDL97, KJJ99]) against secure device, they are more likely to compromise the long-term key (which attracts the attackers mostly) rather than ephemeral key. Since such attacks might be very expensive.
2. \( A \) may issue a polynomial number of queries as described in the adversarial model, namely \( A \) may make queries: Send, StateReveal, Corrupt, EstablishParty and RevealKey. But the EstablishParty query is not allowed to be asked by the adversary while formulating mCK security as Definition 25.\(^4\)

3. At some point, \( A \) may issue a Test\((\pi^s_i)\) query to an oracle \( \pi^s_i \) during the game with only once.

4. At the end of the game, \( A \) may terminate with outputting a bit \( b' \) as its guess for \( b \) of Test query.

### 3.3 Secure Two Party AKE Protocols

In this section, we are going to present the security notions for specific class of protocols via the modified CK (mCK) security definition and eCK security definitions. In mCK security, we define two main security goals for 2AKE: (i) the protocol is a secure authentication protocol, and (ii) the protocol is a secure key exchange protocol, thus an adversary cannot distinguish the real session key from a random key. In the latter we present a stronger security notion i.e. eCK security for one-round key exchange protocols, which formalizes only security of key distribution.

For the mCK security definition, we first need the notion of freshness to exclude trivial attacks in our model.

**Definition 24** (mCK-Freshness). Let \( \pi^s_i \) be accepted oracle (i.e. \( \Phi^s_i = \text{accept} \)) with intended partner \( \text{ID}_j \in \text{pid}^s_i \) \( (j \neq i) \). Let \( \pi^t_j \) be an oracle (if it exists), such that \( \pi^s_i \) and \( \pi^t_j \) have matching sessions. Then the oracle \( \pi^s_i \) is said to be mCK-fresh if none of the following conditions holds:

- \( \mathcal{A} \) queried Corrupt\((\text{ID}_j)\) prior to \( \Phi^s_i = \text{accept} \).
- \( \mathcal{A} \) queried either StateReveal\((\pi^s_i)\) or RevealKey\((\pi^s_i)\).
- if \( \pi^t_j \) exists, \( \mathcal{A} \) queried either StateReveal\((\pi^t_j)\) or RevealKey\((\pi^t_j)\).

Now we define the security of a 2AKE protocol with entity authentication and key distribution in presence of mCK adversaries.

**Definition 25** (mCK Security). We say that an adversary \( \mathcal{A} \) \((t, \epsilon)\)-breaks the mCK security of a correct two party AKE protocol \( \Sigma \), if \( \mathcal{A} \) runs the AKE security game within time \( t \), and at least one of the following conditions holds:

1. When \( \mathcal{A} \) terminates, with probability at least \( \epsilon \) there exists an oracle \( \pi^s_i \), such that

\(^4\)We don’t allow the EstablishParty query in the mCK model, since the security of our compilers in Chapter 4 against CIDPK attacks is unclear.
3 Security Models for Authenticated Key Exchange

- \( \pi_i^s \) has internal state \( \Phi_i^s = \text{accept} \), and
- \( \pi_i^s \) is mCK-fresh, and
- for party \( \text{ID}_j \in \text{pid}_i^s (j \neq i) \) there is no unique oracle \( \pi_j^t \) such that \( \pi_i^s \) and \( \pi_j^t \) have matching sessions.

2. If a \text{Test} query has been issued to an oracle \( \pi_i^s \) and \( \pi_i^s \) is mCK-fresh, then the probability that the bit \( b' \) returned by \( \mathcal{A} \) equals to the bit \( b \) chosen by the \text{Test} query satisfies

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

We say that a correct two party AKE protocol \( \Sigma \) is \((t, \epsilon)\)-mCK-secure, if there exists no adversary that \((t, \epsilon)\)-breaks the mCK security of \( \Sigma \).

Most of the one-round 2AKE protocols can provide eCK security. Note that it is impossible for any one-round 2AKE protocol to achieve mutual authentication in two moves that is why we only consider the session key security.

**Definition 26 (eCK-Freshness).** Let \( \pi_i^s \) be an accepted oracle with intended partner \( \text{ID}_j \in \text{pid}_i^s (j \neq i) \). Let \( \pi_j^t \) be an oracle (if it exists), such that \( \pi_i^s \) has a matching session to \( \pi_j^t \). Then the oracle \( \pi_i^s \) is said to be eCK-fresh if none of the following conditions holds:

1. \( \mathcal{A} \) queried \text{EstablishParty}(\text{ID}_j, pk_{\text{ID}_j}, pf_{\text{ID}_j}) \).
2. \( \mathcal{A} \) queried \text{RevealKey}(\pi_i^s) \).
3. \( \mathcal{A} \) queried both \text{Corrupt}(\text{ID}_i) and \text{StateReveal}(\pi_i^s) \).
4. if \( \pi_j^t \) exists, \( \mathcal{A} \) queried \text{RevealKey}(\pi_j^t) \).
5. If \( \pi_j^t \) exists, \( \mathcal{A} \) queried both \text{Corrupt}(\text{ID}_j) and \text{StateReveal}(\pi_j^t) \).
6. If \( \pi_j^t \) does not exist, \( \mathcal{A} \) queried \text{Corrupt}(\text{ID}_j) \).

**Definition 27 (eCK Security).** We say that an adversary \( \mathcal{A} \) \((t, \epsilon)\)-breaks the eCK security of a correct two party AKE protocol \( \Sigma \), if \( \mathcal{A} \) runs the AKE security game within time \( t \), and the following condition holds:

- If a \text{Test} query has been issued to an oracle \( \pi_i^s \) and \( \pi_i^s \) is eCK-fresh, then the probability that the bit \( b' \) returned by \( \mathcal{A} \) equals to the bit \( b \) chosen by the \text{Test} query satisfies

\[
| \Pr[b = b'] - 1/2 | > \epsilon,
\]

We say that a correct two party AKE protocol \( \Sigma \) is \((t, \epsilon)\)-eCK-secure, if there exists no adversary that \((t, \epsilon)\)-breaks the eCK security of \( \Sigma \).
3.4 Secure Group AKE Protocols

For the security definition, we need the notion about the freshness of oracles which formulates the restrictions on the adversary with respect to performing these queries addressed in the execution environment.

**Definition 28 (g-eCK Freshness).** Let $\pi_i^s$ be an accepted oracle. Let $\pi_S = \{\pi_j^s\}_{ID_j \in \text{pid}_i^s, j \neq i}$ be a set of oracles (if they exist), such that $\pi_i^s$ has a matching session to $\pi_j^s$ for each $ID_j \in \text{pid}_i^s$. Then the oracle $\pi_i^s$ is said to be g-eCK-fresh if none of the following conditions holds:

1. $A$ queried $\text{EstablishParty}(ID_j, pk_{ID_j}, pf_{ID_j})$ with some $ID_j \in \text{pid}_i^s$.
2. $A$ queried $\text{RevealKey}(\pi_i^s)$.
3. $A$ queried both $\text{Corrupt}(ID_i)$ and $\text{StateReveal}(\pi_i^s)$.
4. For some oracle $\pi_j^s \in \pi_S$, $A$ queried $\text{RevealKey}(\pi_j^s)$.
5. For some oracle $\pi_j^s \in \pi_S$, $A$ queried both $\text{Corrupt}(ID_j)$ and $\text{StateReveal}(\pi_j^s)$.
6. If $ID_j \in \text{pid}_i^s$ ($j \neq i$) and there is no oracle $\pi_j^s$ such that $\pi_i^s$ has a matching session to $\pi_j^s$, $A$ queried $\text{Corrupt}(ID_j)$.

Security of GAKE protocols is now defined by requiring that the protocol is a session key secure, thus an adversary cannot distinguish the session key from a random key.

**Definition 29 (g-eCK Security).** We say that an adversary $A (t, \epsilon)$-breaks the g-eCK security of a correct group AKE protocol $\Sigma$, if $A$ runs the AKE security game within time $t$, and the following condition holds:

- If a $\text{Test}$ query has been issued to an oracle $\pi_i^s$ and $\pi_i^s$ is g-eCK-fresh, then the probability that the bit $b'$ returned by $A$ equals to the bit $b$ chosen by the $\text{Test}$ query is bounded by

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

We say that a correct group AKE protocol $\Sigma$ is $(t, \epsilon)$-g-eCK-secure, if there exists no adversary that $(t, \epsilon)$-breaks the g-eCK security of $\Sigma$.

3.5 Remarks on Models

We leave out the definition of session identifier to each protocol itself, instead we only specify the minimum requirements on $\text{sid}$ in models, i.e. the defined $\text{sid}$ is required to be unique at each party and is generated on-the-fly. This approach basically follows the idea on formulating partnership in the model [BPR00] introduced by Bellare, Pointcheval.
and Rogaway. We didn’t adopt a general approach to define partnership based on the full communication transcript like the matching conversations [BR94] or the sid defined in the first eCK model [LLM07], since based on such kind of sid there might exist trivial theoretical attacks on specific protocols in certain strong models, see the detail example in Section 4.5. On the other hand, we are constructing new protocols, and it is easy to define session identifier within each protocol independently. In a nutshell, the approach utilizing ‘full communication transcript’ for defining session identifier is not suitable for all protocols in a strong security model where KCI attacks are formulated or leakage of session states are allowed. Therefore one should be very careful on defining session identifier for specific protocol. Our session identifier definition also enables us to formulate a unified execution environment for these security models.

Moreover, a protocol of course could define session identifier including all protocol messages and then this may imply the partnership defined based on the full communication transcript. The role (e.g. the initiator and responder) was not considered in our partnership definition, because we would like to make it to be compatible with group AKE. In a group AKE protocol, it is hard to determine the role of each party. On the other hand it is very easy to plug the role related information into the session identifier definition for two party AKE. For instance, one could sort the messages used to define session identifier with role (say putting the messages of initiator in front of the responder’s). In order to ‘correctly’ provide the entity authentication security property in the mCK model, we further stress that a protocol with explicit entity authentication should at least exclude the message of last protocol pass while defining session identifier. As otherwise the adversary can simply drop the last message to make an oracle accept without matching session.

We summarize the security goals and attributes that are covered by the models used in this thesis in Table 3.1. The mCK security and eCK security are different from the security goal and freshness definition (in particular for StateReveal, EstablishParty and Corrupt queries). For example, the mCK security formalized entity authentication in contrast to eCK model, whereas the latter allows adversaries to reveal states of target session. We need two distinct security notions since we have to consider the strong security for specific protocols independently which are only provably secure in corresponding models, e.g. a one-round protocol might be provably secure only in the eCK model nor in the mCK model since it is unable to achieve explicit entity authentication in two protocol pass. In addition, the g-eCK model might imply the eCK model when the group size equals two. But we define two security definitions separately for better understanding.
### 3 Security Models for Authenticated Key Exchange

<table>
<thead>
<tr>
<th>Security goals</th>
<th>Security attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAuth</td>
<td>KD</td>
</tr>
<tr>
<td>mCK as Def. 25</td>
<td>√</td>
</tr>
<tr>
<td>EAuth</td>
<td>KSK</td>
</tr>
<tr>
<td>mCK as Def. 25</td>
<td>√</td>
</tr>
<tr>
<td>EAuth</td>
<td>KCI</td>
</tr>
<tr>
<td>mCK as Def. 25</td>
<td>×</td>
</tr>
<tr>
<td>EAuth</td>
<td>CIDPK</td>
</tr>
<tr>
<td>mCK as Def. 25</td>
<td>√</td>
</tr>
<tr>
<td>EAuth</td>
<td>FFS</td>
</tr>
<tr>
<td>mCK as Def. 25</td>
<td>√</td>
</tr>
<tr>
<td>EAuth</td>
<td>wFFS</td>
</tr>
<tr>
<td>mCK as Def. 25</td>
<td>×</td>
</tr>
<tr>
<td>EAuth</td>
<td>Target LSS</td>
</tr>
<tr>
<td>mCK as Def. 25</td>
<td>×</td>
</tr>
<tr>
<td>EAuth</td>
<td>Other LSS</td>
</tr>
<tr>
<td>mCK as Def. 25</td>
<td>√</td>
</tr>
</tbody>
</table>

**Table 3.1:** Summary on models used in this thesis.

The terms ‘EAuth’ and ‘KD’ are abbreviations of entity authentication and key distribution. The term ‘Target LSS’ denotes the leakage of session states from target session and its partner session, and the term ‘Other LSS’ denote the leakage of session states from sessions other than test session and its partner session. The symbols ‘√/×’ means that the model does/doesn’t capture the attack.
4 New Modular Compilers for Authenticated Key Exchange

In this chapter, we present three 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 reveal state information of the protocol participants. Surprisingly, 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. Our compilers are not only a useful tool for the design of new AKE systems in a modular and less error-prone fashion, but they also help to relax the assumptions on existing, practical key exchange mechanisms which are not known to be provably secure AKE protocols. In contrast to previous results, we do not require that key computed by the KE protocol is handed over to the compiler what helps to avoid additional and costly modifications of existing KE-based systems.

Paper. This Chapter is based on the joint works with Yong Li and Sven Schaege which is currently in submission. Yong Li, Sven Schaege and I figured out together the constructions and security analysis of the second compiler in Section 4.3 and the third construction in Section 4.4. In addition, I discovered a new theoretical attack called RAP attack that may be applicable to an AKE protocol while it is analysed in certain strong security model. Yong and I found out some previous works unfortunately overlooked this attack that may trivially invalid their security results, see detail about the RAP attack in Section 4.5. My other contributions to this research focus on the construction of the first compiler and its security analysis in Section 4.2.

4.1 Background

4.1.1 Motivations

In many existing systems both of these tasks are addressed by a single protocol. This can yield very efficient solutions. However, there are several 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 work, 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 AKEs. 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 AKEs which are specifically crafted to fit to 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 analyzed 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 AKEs. 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.

4.1.2 Contributions

We present three very efficient compilers that construct secure AKEs from authentication protocols and passively secure key exchange protocols. To the best of our knowledge, they are the first that are efficient and truly generic, i.e. they do not require any modifications in the underlying KE and AP protocols. They are thus also 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 did require costly modifications on the key exchange protocol such that either the messages have to be modified or the secret $k$ also has to be output to the compiler. 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 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 perfect 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. 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. For encrypted key transport only a single additional move is required (as there is only one party which relies on public keys). Our restriction is useful to design protocols with perfect forward secrecy, which states that even after the compromise of long-term keys previously executed sessions remain secure. It is, for example very easy to see that if the secret (long-term) key in classical encrypted key transport (where party A sends to B a randomly drawn session key which is encrypted with B’s public key) is corrupted by the adversary, the attacker can easily open the ciphertext of all the previous session keys and reveal them. The same restriction is made on the KE protocols which are used in the recent compiler by Jager, Kohlar, Schäge, and Schwenk (JKSS) [JKSS10b]. The well-known compiler by Katz and Yung (KY) uses a slightly different approach by directly requiring that the input protocol provides perfect forward secrecy [KY03]. We stress that it is very easy to check whether a KE protocol does actually fall into this class; one just inspects if the session key computation involves any long-term secret. This can also be done for existing protocols which are complex and where a security proof of AKE is not known, for example when applying our compilers to relax the security assumptions of existing key exchange based applications.

We present three compilers each of which relies on a different authentication mechanism. Our first compiler is very efficient. It relies on signature schemes and only requires

\footnote{This restriction is implicit in [JKSS10b] (JKSS). An explicit statement can be found in the full version of the paper [JKSS10a].}
two additional moves in which signatures are exchanged. The second compiler relies on public key encryption systems, one-time message authentication code (OTMAC) scheme, and collision resistant hash functions (CRHFs). 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. Previous compilers (like [BCK98]) have used combinations of public key encryption (PKE) systems and regular message authentication codes (MACs) while the latter are likely to introduces additional complexity assumptions or rely on more inefficient algebraic constructions. In contrast, it is well-know how to construct more efficient one-time message authentication code (OTMAC) based on pairwise independent hash function [WC81]. We use CRHFs solely for the purpose of domain extension. In the latter we present a variant of our second compiler which relies on a combination of a key encapsulation mechanism (KEM) and a regular MAC. In scenarios where the MAC computations are very cheap this variant might be more efficient as KEMs in general are more efficient than PKEs (as KEMs only have to securely ‘encrypt’ uniformly random messages in contrast to PKEs which have to work for any message distribution).

In this work, we focus on practicality and efficiency of our solutions. We stress that we could generalize our compiler in a straight-forward way to also 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. It is easy to extend the original definition to also cover stateful verifiers. In Chapter 2 we present a modified definition of TBAS that also cover verifiers with secret states. This may be of independent interest. In terms of efficiency, the recent JKSS compiler is less efficient than any of our compilers as it additionally requires two random nonces and two MAC values to be exchanged. All our solutions work in the standard model, i.e. without assuming random oracles.

Technical Contributions. Our efficiency improvements rely on two 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) 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.

In particular we identify new problems when defining AKE security based on different partnership definitions that also models KCI attacks and leakage of secret states. Specifically, we present a new theoretical attack called randomized authentication primitive (RAP) attack that may be applied to AKE protocols which are constructed with randomized authentication primitive and are proved in a strong model (formulated KCI attacks and leakage of secret states) where the partnership is defined involving the message generated by underlying randomized authentication primitive. We show a concrete attack example based on our first signature-based compiler (and signature-based JKSS compiler) that may be subject to this attack in the mCK model if the session identifier is defined inappropriately. However this attack does not break the protocol in any practically harmful way but would invalid the security proof in corresponding model. In addition, we somehow generalize the attack idea. Our attack can be applied to several protocols, which previously have been believed to be provably secure in corresponding security model. Several solutions on how to avoid the RAP attacks are also given in this thesis, that might be not only useful for these problematic existing protocols to retain provably security but also a guideline for designing new protocols. One could utilize deterministic scheme as alternative, or define the partnership without the messages generated by randomized authentication primitive. In a nutshell, we solved the problem on how to construct probably secure generic AKE compilers from AP protocols and passively secure KE protocols in presence of strong adversaries which are provided with extended capabilities when modelling KCI attacks and leakage of secret states.

4.1.3 Related Works for AKE Compilers

In 1998, Bellare, Canetti and Krawczyk (BCK) were the first to consider a modular way for the development of AKEs [BCK98]. 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 \( \hat{A} \) needs to transmit to \( \hat{B} \) there will be some additional communication with \( \hat{B} \) in which \( \hat{B} \) sends a random nonce to \( \hat{A} \) and \( \hat{A} \) 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 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 [KY03]. 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. When restricted to the two-party case, this compiler is much more efficient in terms of protocol moves as, 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 [BD94] 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 perfect forward secrecy. However, it does require that the input group key protocols already provide perfect 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). We admit that it is easy to construct an artificial and rather impractical protocol which does rely on long-term keys but which does not provide perfect forward secrecy. However, to the best of our knowledge all existing long-term key based KE protocols use the long-term key in such a way that the corruption of the long-term key would violate the security of previous sessions. Intuitively, our approach implies perfect 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. So, there is no value re-used in several sessions that can, when revealed, provide information on the keys of distinct sessions. 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 [JKSS10b], which is proved secure in a BR model. In terms of efficiency, this compiler is closest to our results. Basically, the compiler, after executing the KE protocol, makes \( \hat{A} \) and \( \hat{B} \) additionally exchange 1) random nonces, 2) signatures over these nonces and the KE transcript and 3) two MAC values which have been computed over all the previous messages using the KE key. In the last step the compiler requires the key generated by the KE to be available to the compiler. As mentioned above this compiler is less efficient than our solution. At the same time it does not preserve security in our strong security model (all of the above compilers do not consider state reveals). In Section 4.1.4 we present an attack that is applicable to the JKSS compiler if we additionally allow state reveals. (We stress that the JKSS security model does actually not consider such adversarial capabilities. However, the attack suffices to show that our compiler guarantees security even in the presence of much stronger adversaries.) In [JKSS10b] the authors also consider a second compiler which is more efficient than the first but only works in the random oracle model. As with the first JKSS compiler this second compiler also requires the session key of the underlying key exchange protocol as input. The second compiler is formulated more abstractly and it is claimed that it works with general challenge and response based authentication protocols. Remarkably, when instantiated with a signature-based solution, this compiler is slightly less efficient than our signature-based variant as additionally random nonces have to be exchanged between the parties. However, we stress that our results are secure in the standard model, i.e. without relying on random oracles.

More recently, Boyd et al. [BG11] and Cremers et al. [CF12] proposed two compilers respectively, which are referred to as Compiler-MAC [BG11] and Compiler-SIG [CF12] in this work. These two compilers all aimed to compile one-round two party AKE protocols with weak perfect forward secrecy to achieve perfect forward secrecy without changing the internal execution of original protocols. However, they need very strong assumptions on the compiled protocols, i.e. they should be proved secure in either CK model [CK01] or eCK model [LLM07]. Please note that our works have different motivations from theirs. Because again we aim to compile protocols which are not known to be AKE secure and we only require the compiled protocols to be passively secure. Moreover, we also show that both Compiler-MAC and Compiler-SIG are problematic, they are subject to our new attack (see detail in Section 4.5).

4.1.4 The Attack due to Leakage of Session States from KE Protocol

In this section, we present an attack on JKSS compiler [JKSS10b] caused by leakage of session states from KE protocol. We assume that the JKSS protocol runs in the pre-specified peer setting [CK02] and no identities have been exchanged in the key exchange protocol. Note that the JKSS compiler does not provide explicit information on whether
they use the post-specified peer setting. Because there is no identity related information appeared in JKSS protocol description.

The attack scenario requires the adversary to be capable of the strong StateReveal capability to obtain the secret state of the KE session. We assume that the session key returned by the KE is part of the secret state in the JKSS compiler. After obtaining the KE key $k$ from the KE protocol instance, the adversary is able to compute the key $K_{MAC}$ herself, as well as the subsequent MACs. Meanwhile, the adversary can generate the signatures on behalf of any corrupted party. The attack on the responder $B$ proceeds as follows, which is also depicted in Figure 4.1:

**Figure 4.1:** Attack on JKSS AKE Compiler due to Leakage of States

1. The adversary either corrupts a party’s long-term key. Let the corrupted party be $E$ having long-term key pair $(pk_E, sk_E)$.

2. After executing a KE exchange protocol instance between oracles $\pi^{s}_{A,B}$ and $\pi^{t}_{B,E}$ with transcript $T_{KE}$, $E$ obtains the key $k$ from $\pi^{t}_{B,E}$ via the StateReveal($\pi^{t}_{B,E}$) query and computes the values $K_{MAC} = PRF(k, "MAC")$ and $K = PRF(k, "ENC")$ herself.

3. $E$ honestly relays the nonce $r_A$ from $\pi^{s}_{A,B}$ to $\pi^{t}_{B,E}$ and the nonce $r_B$ from $\pi^{t}_{B,E}$ to $\pi^{s}_{A,B}$.

4. $E$ computes the signature $\sigma_E = SIG.Sign(sk_E, T_1)$ where $T_1 = T_{KE}|r_A|r_B$, and
sends $\sigma_E$ to $\pi^s_{B,E}$. $E$ honestly relays the signature $\sigma_B = \text{SIG}_E^{\text{Sign}}(sk_B, T_1)$ from $B$ to $\pi^s_{A,B}$.

5. $E$ computes the $w^E_B = \text{MAC}(K_{MAC}, T_2)$ where $T_2 = \sigma_A || \sigma_B$, and sends $w^E_B$ to $\pi^s_{A,B}$.

6. Finally, $\pi^s_{A,B}$ accepts but there is no oracle of $B$ having matching conversation to oracle $\pi^s_{A,B}$. The oracle $\pi^s_{A,B}$ accepts the signature $\sigma_B$ since it is a correct signature of $B$ on transcript $T_1$, and accepts the $w^E_B$ computed by $E$ since the adversary has computed it using the derived key which is the same as $K_{MAC}$ of $\pi^s_{A,B}$. Then the adversary can select oracle $\pi^s_{A,B}$ to win the game.

Note that the above attacks can also be applied to the second JKSS compiler with minimal modification.

### 4.2 Compiler for AKE Protocol from Signature

In this section we present an efficient generic compiler that turns passively secure KE protocols as defined to protocols which fulfil the security guarantees formulated by mCK model. Concerning the identities of session participants, we have the following cases:

- **Case 1**: the identities have been exchanged in the KE protocol instance;
- **Case 2**: no identities have been exchanged in the KE protocol instance but the authentication protocol is executed in the pre-specified peer setting [CK01, MU08];
- **Case 3**: no identities have been exchanged in the KE protocol instance but the authentication protocol is executed in the post-specified peer setting;

We assume without loss of generality that our compiled protocol either is a Case 1 or a Case 2 protocol. If it is a Case 3, we can easily spend another initial extra round to exchange the identities of the participants.

#### 4.2.1 Protocol Description

The compiler takes as input the following building blocks:

- A passively secure key exchange protocol KE without long-term key,
- a digital signature scheme $\text{SIG} = (\text{SIG}_E^{\text{Gen}}, \text{SIG}_E^{\text{Sign}}, \text{SIG}_E^{\text{Vfy}})$.

To set up the system, the public system parameter $\text{pms} := \text{pms}^{ke}$ may be generated by running $\text{pms}^{ke} \leftarrow \text{KE}_{\text{Setup}}(1^k)$. Each party $\hat{A}$ is assumed to possess a pair of long-term keys generated as $(sk_{\hat{A}}, pk_{\hat{A}}) \leftarrow \text{SIG}_{\text{Gen}}(1^k)$. In the sequel, we will use the
Protocol Execution: the compiled protocol between two parties \( \hat{A} \) and \( \hat{B} \) proceeds as follows, which is also informally depicted in Figure 4.2.

1. \( \hat{A} \) and \( \hat{B} \) run the key exchange protocol. Throughout this protocol run, both parties compute the key \( k \) and record the transcript as \( T_{KE}^{\hat{A}} \) and \( T_{KE}^{\hat{B}} \), where \( T_{KE}^{\hat{B}} \) consists of the list of all messages sent and received by party \( \hat{D} \in \{ \hat{A}, \hat{B} \} \). Each party \( \hat{D} \in \{ \hat{A}, \hat{B} \} \) sets the session identifier as \( \text{sid} = T_{KE}^{\hat{D}} \).

2. \( \hat{A} \) sets \( T_{1}^{\hat{A}} := T_{ke}^{\hat{A}} || \hat{A} || T_{ke}^{\hat{B}} \), and \( \hat{B} \) sets \( T_{1}^{\hat{B}} := T_{ke}^{\hat{B}} || \hat{A} || T_{ke}^{\hat{B}} \). \( \hat{A} \) computes a signature \( \sigma_{\hat{A}} := \text{SIG.Sign}(sk_{\hat{A}}, "1" || T_{1}^{\hat{A}}) \) under \( \hat{A} \)'s secret key \( sk_{\hat{A}} \) and send it to \( \hat{B} \). Meanwhile, \( \hat{B} \) computes a signature \( \sigma_{\hat{B}} := \text{SIG.Sign}(sk_{\hat{B}}, "2" || T_{1}^{\hat{A}}) \) under \( \hat{B} \)'s secret key \( sk_{\hat{B}} \) and send it to \( \hat{A} \).

3. Upon receiving signature on each side, \( \hat{A} \) accepts if and only if \( \text{SIG.Vfy}(pk_{\hat{B}}, "2" || T_{1}^{\hat{A}}, \sigma_{\hat{B}}^{\hat{A}}) = 1 \). \( \hat{B} \) accepts if and only if \( \text{SIG.Vfy}(pk_{\hat{A}}, "1" || T_{1}^{\hat{B}}, \sigma_{\hat{A}}^{\hat{B}}) = 1 \).

**Implementation and Session States:** In the concrete protocol implementation, we specify that only the signature generation operations need to be done on secure device, i.e. the step 2 in the above protocol description. We thus assume only the ephemeral secret \( esk \) used in each KE protocol instance will be stored in the variable \( st \). That is, we do not assume that the \text{StateReveal} query reveals any information about the authentication mechanism that involves the long-term key. This is consistent with the usual distinction (which is made formal by providing different types of queries) between the compromise of long-term keys via the \text{Corrupt} query on the one hand and the session keys via the \text{RevealKey} query on the other hand. The KE protocol (which only computes data that needs to be
kept secret for a single session) in contrast, is in practice usually not protected by trusted devices.

### 4.2.2 Security Analysis

In the proof we will essentially rely on Lemma 1 that applies to key exchange protocol without long-term keys. Lemma 1 shows that if both parties execute the protocol in an honest way, then collisions of the ephemeral public keys can be excluded and the transcript is unique. This is directly useful in the key indistinguishability game. However, it also useful in the authentication part of the AKE definition (Property 1 of Definition 25) although we do only concentrate on one party. Let us elaborate on this. One could assume that Lemma 1 does not apply to the authentication game as the adversary can send arbitrary ephemeral keys to the oracle \( \pi_s^i \). It is now not guaranteed that the ephemeral public keys computed by \( \pi_s^i \) are unique and thus that the transcript is unique. However, the ephemeral public keys sent by the adversary need to be authenticated by some uncorrupted party \( ID_j \). Otherwise \( \pi_s^i \) will not accept. This means that the only way of the adversary to win is to replay the authentication values \( epk \) of some other session with \( ID_j \). However, these authentication value are also over the ephemeral public keys sent in that session data. This means that if the authentication mechanism is secure and \( ID_j \) is uncorrupted (i.e. was uncorrupted before) than the adversary can only win the authentication game if it also replays the ephemeral public keys \( epk \) of that session. However, \( epk \) has been generated honestly and thus the lemma applies.

**Theorem 1.** Assume that the KE protocol without long-term key is \((t, \epsilon^{KE})\)-passively secure, and the signature scheme is \((q_{sig}, t, \epsilon^{SIG})\)-secure against existential forgeries under adaptive chosen-message attacks, then the above protocol is a \((t', \epsilon)\)-mCK-secure AKE protocol in the sense of Definition 25 with \( t' \approx t \), and \( q_{sig} \geq \rho \), and it holds that

\[
\epsilon \leq 2\ell \cdot \epsilon^{SIG} + \rho(\rho\ell + 2) \cdot \epsilon^{KE}.
\]

We prove Theorem 1 in two stages. First, we show that the AKE protocol is a secure authentication protocol except for probability \( \epsilon_{auth} \), that is, the protocol fulfills security property 1.) of the AKE definition. Generally speaking, the authentication property is guaranteed by the security of the digital signatures scheme. If an uncorrupted oracle \( \pi_i^s \) with internal state \( \Phi_i^s = \text{accept} \) with intended communication partner \( ID_j \), where \( ID_j \) is uncorrupted, then the signature value received by oracle \( \pi_i^s \) was generated and sent by its partner oracle \( \pi_j^t \) (which has matching session to \( \pi_i^s \)), otherwise the adversary must have forged a signature on behalf of party \( ID_j \). In the next step, we show that the session key of the AKE protocol is secure except for probability \( \epsilon_{ind} \) in the sense of the Property 2.) of the AKE definition. The security of the authentication protocol guarantees that there can only be passive attackers on the test oracle, so that we can
conclude security from the security of the underlying KE protocols. We require that
$q_{\text{sig}} \geq \rho$, since we may need to correctly simulate all signatures for the $\rho$ oracles of the party that is attacked in the security game. Then we have the overall probability $\epsilon$ that adversary breaking the protocol is at most $\epsilon \leq \epsilon_{\text{auth}} + \epsilon_{\text{ind}}$.

**Lemma 2.** If the KE protocol is $(t, \epsilon_{\text{KE}})$-passively secure, and the signature scheme is $(q_{\text{sig}}, t, \epsilon_{\text{SIG}})$-secure against existential forgeries under adaptive chosen-message attacks, then the above protocol meets the security Property 1. of the mCK security definition except for probability with

$$\epsilon_{\text{auth}} \leq \rho \ell \cdot \epsilon_{\text{KE}} + \ell \cdot \epsilon_{\text{SIG}},$$

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

**Proof.** Let $S_{\delta}^1$ be the event that (i) there exists oracle $\pi_{i}^{s^*}$ reaches internal state $\Phi = \text{accept}$ with intended communication partner $\text{ID}_j$, but (ii) there is no oracle $\pi_{j}^{t}$ such that $\pi_{i}^{s^*}$ and $\pi_{j}^{t}$ have matching sessions, in Game $\delta$.

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

$$\Pr[S_{0}^1] = \epsilon_{\text{auth}}.$$

**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 $\text{abort}_{\text{eph}}$ and aborts, if during the simulation an ephemeral key $\text{epk}_{i}^{s}$ is computed by an oracle $\pi_{i}^{s}$ but it has been sampled by another oracle $\pi_{w}^{s}$ 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 $\text{KE.EKGen}$ and the probability $\epsilon_{\text{KE}}$. Since there are $\rho \ell$ oracles at all each of which would execute one general $\text{KE.EKGen}$, therefore the event $\text{abort}_{\text{eph}}$ occurs with probability $\Pr[\text{abort}_{\text{eph}}] \leq \rho \ell \cdot \epsilon_{\text{KE}}$. We have that

$$\Pr[S_{0}^1] \leq \Pr[S_{1}^1] + \rho \ell \cdot \epsilon_{\text{KE}}.$$

**Game 2.** This game proceeds exactly as before, but raises event $\text{abort}_{\text{sig}}$ and aborts if the following condition holds: there exists a mCK-fresh oracle $\pi_{i}^{s}$ such that $\pi_{i}^{s}$ 'accepts', and

- $\pi_{i}^{s}$ has transcript $T_{1}^{i,s} = T_{\text{KE}}^{i,s}||\text{ID}_i||\text{ID}_j$.
- there is no unique oracle $\pi_{j}^{t}$ which has the transcript $T_{1}^{j,t} = T_{\text{KE}}^{j,t}||\text{ID}_i||\text{ID}_j$, such that $T_{1}^{i,s} = T_{1}^{j,t}$.
- but the signature received by $\pi_{i}^{s}$ that is computed over "1" || $T_{1}^{i,s}$ (resp. "2" || $T_{1}^{i,s}$) verifies correctly under the long-term public key $pk_{\text{ID}_j}$.
We have
\[ \Pr[S_1] \leq \Pr[S_2] + \Pr[\text{abort}_{\text{sig}}]. \]

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

When \( \text{abort}_{\text{sig}} \) is raised, then this means that the adversary has forged a signature on behalf of an uncorrupted party \( \text{ID}_j \). If the simulator guessed the party \( \text{ID}_j \) that the adversary attacked (such that \( \theta = j \)) correctly, which happens with probability \( 1/\ell \), then the \( F \) can use the signature received by \( \pi^*_i \) to break the EUF-CMA security of the underlying signature scheme with success probability \( \epsilon_{\text{SIG}} \). So the event \( \text{abort}_{\text{sig}} \) happens with the probability \( \frac{\Pr[\text{abort}_{\text{sig}}]}{\ell} \leq \epsilon_{\text{SIG}} \). Therefore we have
\[ \Pr[S_1] \leq \Pr[S_2] + \ell \cdot \epsilon_{\text{SIG}}. \]

Please note each oracle contributes at least one unique ephemeral key, which is included in the transcript of \( \text{KE} \) instance. And all identities are distinct. Hence, in Game 2 each accepting mCK-fresh oracle \( \pi^*_i \) has a unique ‘partner’ oracle \( \pi^*_j \) sharing the same transcript \( T_1 \). Therefore we have \( \Pr[S_3] = 0 \).

Sum up probabilities from Game 0 to Game 2, we proved this lemma.

\[ \square \]

**Lemma 3.** If the KE protocol is \((t, \epsilon_{\text{KE}})\)-passively secure, the signature scheme is \((q_{\text{sig}}, t, \epsilon_{\text{SIG}})\)-secure against existential forgeries under adaptive chosen-message attacks, then for any adversary running in time \( t \), the probability of \( A \) to correctly answer the Test-query is at most \( 1/2 + \epsilon_{\text{ind}} \) with
\[ \epsilon_{\text{ind}} \leq \ell \cdot \epsilon_{\text{SIG}} + \rho \ell (\rho \ell + 1) \cdot \epsilon_{\text{KE}}, \]
where all quantities are as the same as stated in the Theorem 2.

**Proof.** Let \( S_3^\delta \) denotes the event that the \( A \) correctly guesses the bit \( b \) sampled by the Test-query in Game \( \delta \). Let \( \text{Adv}_\delta := \Pr[S_3^\delta] - 1/2 \) denote the advantage of \( A \) in Game \( \delta \).

Let oracle \( \pi^*_i \) denote the fresh test oracle and let \( \pi^*_j \) denote the oracle having matching session to \( \pi^*_i \). Consider the following sequence of games.
4 New Modular Compilers for Authenticated Key Exchange

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

\[ \Pr[S_0] = \epsilon_{ind} + 1/2 = \text{Adv}_0 + 1/2. \]

**Game 1.** The challenger in this game proceeds as before, but it aborts if the test oracle accepts without unique partner oracle. Thus we have

\[ \text{Adv}_0 \leq \text{Adv}_1 + \epsilon_{auth} \leq \text{Adv}_1 + \rho \ell \cdot \epsilon_{KE} + \ell \cdot \epsilon_{SIG}, \]

where \( \epsilon_{auth} \) is an upper bound on the probability that there exists an oracle that accepts without unique partner oracle in the sense of Definition 25 (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 sessions. Since there are \( \ell \) honest parties and \( \rho \) oracles for each party, the probability that the adversary guesses correctly is at least \( 1/(\rho \ell)^2 \). Thus we have that

\[ \text{Adv}_1 \leq \rho^2 \ell^2 \cdot \text{Adv}_2. \]

**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 \( \tilde{k}^* \). Note that the KE protocol instance executed between the test oracle and its partner oracle only allows for passive adversaries due to Game 1. If there exists an adversary \( A \) which can distinguish this game from the previous game, then we use it to construct an algorithm \( B \) to break the passive security of key exchange protocol as follows. Assume that the adversary \( B \) interacts with the challenger \( C_{KE} \) via \textbf{Execute} query which simulates the passive security game as in the security definition of key exchange. More specifically, \( B \) simulates the challenger in this game for \( A \) which is illustrated as follows:

- At the beginning of the game, \( B \) implements the collection of oracles \( \{ \pi_i^{s*} : i \in [\ell], s \in [\rho] \} \). All long-term public/private key pairs \( (pk_{ID_i}, sk_{ID_i}) \) for each honest entity \( i \) are generated honestly. The adversary \( A \) receives the public keys \( pk_{ID_1}, \ldots, pk_{ID_\ell} \) as input. But note that all long-term keys are useless here for executing KE protocols.

- Meanwhile, \( B \) generates a random ephemeral key vector \( epk_i^s \) and corresponding ephemeral secret vector \( esk_i^s \) for each oracle \( \pi_i^{s*} \) as described in the protocol specification and answers all oracle queries honest except for the test oracle and its partner oracle.
• As for the correctly guessed test oracle $\pi_i^{*}$ and its partner oracle $\pi_j^{*}$, $B$ queries $C_{KE}$ for executing a test protocol instance and obtains $(K_B, T)$ from $C_{KE}$. Otherwise $B$ simulates the test oracles using the transcript $T$ and giving $A K_B$ in return. $A$ may keep asking oracle queries. Meantime, if the ephemeral keys of the test oracle and its partner oracle have been sampled by $B$ before then it can trivially win the KE game and halts.

• Eventually, $B$ returns the bit $b'$ obtained from $A$ to $C_{KE}$.

The simulation of $B$ is perfect since $B$ can always correctly answer all queries from $A$. In particular for those StateReveal queries, $B$ can answer them using those ephemeral secret keys $esk_i^*$ chosen by herself. Meanwhile, $B$ is not required to answer StateReveal, Corrupt or RevealKey queries to test oracle and its partner oracle which should be mCK-fresh, as otherwise the game has been aborted due to the previous games. If $A$ is able to correctly answer the bit $b$ of Test-query with non-negligible probability, so does the adversary $B$ (which breaks the passive security of the KE protocol). Exploiting the security of key exchange protocol, we obtain that

$$Adv_2 \leq Adv_3 + \epsilon_{KE}.$$ 

In this game, the response to the Test query always consists of an uniformly random key, which is independent to the bit $b$ flipped in the Test query. Thus we have $Adv_3 = 0$.

This lemma is proved by putting together of probabilities from Game 0 to Game 3. 

4.3 Compiler for AKE Protocol From PKE and (One-time) MAC

In this section we present a generic compiler to construct AKE protocol without random oracles which mainly uses a public encryption (PKE) scheme and one-time message authentication code (OTMAC) scheme.

4.3.1 Protocol Description

Let us now describe our PKE-OTMAC based AKE compiler. Again we assume that the compiled protocol is a Case 1 or Case 2 protocol.

The compiler takes as input the following building blocks:

• A passively secure key exchange protocol KE without long-term key,

• a public encryption scheme $PKE = (PKE.KGen, PKE.Enc, PKE.Dec)$,
- a collision resistant hash function $\text{CRHF} : \mathcal{K}_{\text{CRHF}} \times \mathcal{M}_{\text{CRHF}} \rightarrow \mathcal{Y}_{\text{CRHF}}$, and
- a message authentication code scheme $\text{MAC} = (\text{MAC.KGen}, \text{MAC.Tag})$, which could be any one-time message authentication code (OTMAC) scheme.

![Figure 4.3: AKE Compiler from PKE and One-time MAC](image)

During the initiation phase, the public system parameter $pms := (pms_{ke}, pms_{pke}, h_{k_{\text{CRHF}}})$ is generated by running $pms_{ke} \leftarrow \text{KE.Setup}(\ell), pms_{pke} \leftarrow \text{PKE.Setup}(\ell)$ and $h_{k_{\text{CRHF}}} \leftarrow \text{CRHF.KGen}(\ell)$. Each party $A$ is assumed to possess a pair of long-term private and public keys generated as $(sk_{A}, pk_{A}) \leftarrow \text{PKE.KGen}(pms_{pke})$.

**Protocol Execution**: The compiled protocol between two parties $A$ and $B$ proceeds as follows, which is also depicted in Figure 4.3.

1. $A$ and $B$ run the key exchange protocol. Throughout this protocol run, both parties calculate the session key $k$ and record the transcripts as $T_{\text{KE}}^{A}$ and $T_{\text{KE}}^{B}$, where...
4. New Modular Compilers for Authenticated Key Exchange

$T_{KE}^D$ consists of the list of all messages sent and received by party $D \in \{\hat{A}, \hat{B}\}$. Each party $D \in \{\hat{A}, \hat{B}\}$ sets the session identifier as $\text{sid} = T_{KE}^D$.

2. $\hat{A}$ sets transcript $T_1^\hat{A} := T_{KE}^\hat{A}||\hat{A}||\hat{B}$ and $\hat{B}$ sets and transcript $T_1^\hat{B} := T_{KE}^\hat{B}||\hat{A}||\hat{B}$.

3. $\hat{A}$ computes a hash value $N_{\hat{A}} := \text{CRHF}(T_1^\hat{A})$ by using the $T_1^\hat{A}$ as input and $\hat{B}$ computes $N_{\hat{B}} := \text{CRHF}(T_1^\hat{B})$.

4. $\hat{A}$ chooses a random key $K_{\hat{A}} \leftarrow \text{MAC.KGen}(1^n)$, and computes a ciphertext $C_{\hat{A}} \leftarrow \text{PKE.Enc}(pk_{\hat{B}}, K_{\hat{A}}||N_{\hat{A}})$ under $\hat{B}$’s public key $pk_{\hat{B}}$ and send it to $\hat{B}$. Meanwhile, $\hat{B}$ chooses a random key $K_{\hat{B}} \leftarrow \text{MAC.KGen}(1^n)$, and computes a ciphertext $C_{\hat{B}} \leftarrow \text{PKE.Enc}(pk_{\hat{A}}, K_{\hat{B}}||N_{\hat{B}})$ under $\hat{A}$’s public key $pk_{\hat{A}}$.

5. Upon receiving the ciphertext $C_{\hat{A}}^B$, $\hat{B}$ sets the transcript $T_2^\hat{B} := T_1^\hat{B}||C_{\hat{A}}^B||C_{\hat{B}}$. $\hat{B}$ decrypts the ciphertext $C_{\hat{B}}^A$, i.e. $K_{\hat{B}}^A||N_{\hat{A}}^B := \text{PKE.Dec}(sk_{\hat{B}}, C_{\hat{B}}^A)$, and rejects if $N_{\hat{A}}^B \neq N_{\hat{B}}$. Moreover, $\hat{B}$ computes $M_{\hat{B}} := \text{MAC.Tag}(K_{\hat{B}}^A, \text{“}2\text{”}||T_2^\hat{B})$ and sends $(M_{\hat{B}}, C_{\hat{B}})$ to $\hat{A}$.

6. Upon receiving messages $(M_{\hat{A}}^A, C_{\hat{A}}^A)$, $\hat{A}$ sets the transcript $T_2^\hat{A} := T_1^\hat{A}||C_{\hat{A}}^A||C_{\hat{B}}^A$ and rejects if $M_{\hat{A}}^B \neq \text{MAC.Tag}(K_{\hat{A}}^B, T_2^\hat{A})$. $\hat{A}$ decrypts the ciphertext $C_{\hat{B}}^A$, i.e. $K_{\hat{B}}^A||N_{\hat{B}}^A := \text{PKE.Dec}(sk_{\hat{B}}, C_{\hat{B}}^A)$, and rejects if $N_{\hat{A}}^A \neq N_{\hat{B}}^A$. Moreover, $\hat{A}$ computes $M_{\hat{A}} := \text{MAC.Tag}(K_{\hat{B}}^A, \text{“}1\text{”}||T_2^\hat{A})$, and sends it to $\hat{B}$. To this end $\hat{A}$ accepts the session.

7. Upon receiving $M_{\hat{A}}^B$, $\hat{B}$ accepts if and only if $M_{\hat{A}}^B = \text{MAC.Tag}(K_{\hat{B}}^A, \text{“}1\text{”}||T_2^\hat{B})$.

Implementation and Session States: We assume the ephemeral secret $esk$ used in each KE protocol instance and the random key $K_{\hat{A}}$ and $K_{\hat{B}}$ used by PKE.Enc (nor the key decrypted by PKE.Dec), will be stored in the variable $st$. This can be achieved via operating the PKE.Dec algorithm and corresponding checks using its output on secure device. For instance the secure device of $\hat{B}$ may perform the following steps: run $K_{\hat{B}}^A||N_{\hat{B}}^A := \text{PKE.Dec}(sk_{\hat{B}}, C_{\hat{B}}^A)$ and return rejection if $N_{\hat{A}}^B \neq N_{\hat{B}}$ or $M_{\hat{B}} \neq \text{MAC.Tag}(K_{\hat{B}}^A, \text{“}2\text{”}||T_2^\hat{B})$.

4.3.2 Security Analysis

In this subsection we prove the security of the compiled AKE protocol in the mCK security model.

Theorem 2. Assume that the KE protocol without long-term key is $(t, \epsilon_{KE})$-passively secure, the PKE scheme is $(q_{pke}, t, \epsilon_{PKE})$-secure against adaptive chosen-ciphertext attacks, and the hash function CRHF is $(t, \epsilon_{CRHF})$-secure and the MAC scheme is $(1, t, \epsilon_{MAC})$-secure against forgeries under adaptive chosen-message attacks. Then the above protocol is a $(t', \epsilon)$-mCK-secure AKE protocol in the sense of Definition 25 with $t' \approx t$, and
4 New Modular Compilers for Authenticated Key Exchange

$q_{pke} \geq \rho$ and holds that

$$
\epsilon \leq 2\epsilon_{\text{CRHF}} + \rho \ell \cdot (2\ell \cdot \epsilon_{\text{PKE}} + 2 \cdot \epsilon_{\text{MAC}} + (\rho \ell + 2) \cdot \epsilon_{\text{KE}}).
$$

We prove this theorem with two lemmas, similar to the proof of Theorem 1.

**Lemma 4.** Assume that the KE protocol is $(t, \epsilon_{\text{KE}})$-passively secure, the PKE scheme is $(q_{pke}, t, \epsilon_{\text{PKE}})$-secure against adaptive chosen-ciphertext attacks, and the hash function CRHF is $(t, \epsilon_{\text{CRHF}})$-secure and the MAC is $(1, t, \epsilon_{\text{MAC}})$-secure against forgeries under adaptive chosen-message attacks. Then the above protocol meets the security property 1.) of the mCK security definition except for probability with

$$
\epsilon_{\text{auth}} \leq \rho \ell \cdot \epsilon_{\text{KE}} + \epsilon_{\text{CRHF}} + \rho \ell \cdot (\ell \cdot \epsilon_{\text{PKE}} + \epsilon_{\text{MAC}}),
$$

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

**Proof.** Let $S_{\delta}^1$ be the event that (i) there exists a mCK-fresh oracle $\pi_i^s$ that reaches the internal state $\Phi_i^s = \text{accept}$ with intended communication partner $\text{ID}_j$, but (ii) there is no oracle $\pi_j^t$ such that $\pi_i^s$ and $\pi_j^t$ have matching sessions, in Game $\delta$.

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

$$
\Pr[S_0^1] = \epsilon_{\text{auth}}.
$$

**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 $\text{abort}_{\text{eph}}$ and aborts, if two oracles output the same ephemeral keys. Thus, the event $\text{abort}_{\text{eph}}$ occurs with probability $\Pr[\text{abort}_{\text{eph}}] \leq \rho \ell \cdot \epsilon_{\text{KE}}$ in terms of Lemma 1. We have that

$$
\Pr[S_0^1] \leq \Pr[S_1^1] + \rho \ell \cdot \epsilon_{\text{KE}}.
$$

So that this game ensure that there is no replay attacks against KE protocol in active game. Also only two oracles having matching sessions will share the same transcript due to the uniqueness of ephemeral key.

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

$$
\Pr[S_1^1] \leq \Pr[S_2^1] + \epsilon_{\text{CRHF}}.
$$

In this game, for example, the adversary cannot establish malicious identities to make two different transcripts map to the same hash value, and then launch an unknown key share attacks. All queries in this game will be answered honestly as in the previous game.
Game 3. This game proceeds as the previous game, but now the challenger aborts if it fails to guess the first mCK-fresh oracle $\pi_1^{s*}$ which accepts without matching session. To this end the challenger draws random $i \in [\ell]$ and $s^* \in [\rho]$. Since there are $\rho \ell$ parties with probability $\geq 1/\rho \ell$ this guess is correct. Thus we have that

$$\Pr[S_2] \leq \rho \ell \cdot \Pr[S_3].$$

In the sequel, we assume that the guess of the challenger is always correct.

Game 4. This game proceeds exactly as before, but we use a random key $\widetilde{K}_i^{s*}$ to compute and verify the hash value $M_j^{s*}$ that is independent of the key $K_i^{s*}$ encrypted in ciphertext $C_i^{s*}$. We do the same modification for the oracle $\pi_t^{s*}$ which shares the same transcript $T_1$ with oracle $\pi_s^{s*}$, 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 adversary $A$ that distinguishes this game from the previous game can be used to break the security of the PKE scheme.

Specifically, we could construct an efficient algorithm $B$ to break the PKE scheme (which is given access to a decryption oracle $\mathcal{D\mathcal{E}}\mathcal{C}$) by running $A$ as a subroutine. When given the public key of the PKE game $pk^*$, $B$ simulates the AKE game for $A$ with answering all queries from $A$ honestly. $B$ firstly guesses an index $j \leftarrow [\ell]$ pointing to the intended communication partner ID$_j$ of the oracle $\pi_i^{s*}$ under attack (with probability at least $1/\ell$) and sets $pk_{ID_j} = pk^*$. Next $B$ sets $pk_{ID_j} = pk^*$ and chooses all other long-term public/private keys honestly. Meanwhile, $B$ computes the ciphertext $C_i^{s*}$ by querying the challenge ciphertext using messages $(\widetilde{K}_i^{s*} || N_i^{s*}, K_i^{s*} || N_i^{s*})$ from the PKE game. As for other oracles of ID$_j$, $B$ queries the $\mathcal{D\mathcal{E}}\mathcal{C}(sk^*, \cdot)$ oracle to obtain the hash keys and to compute the confirmation hash values. Note that if the $C_i^{s*}$ encrypts the key $\widetilde{K}_i^{s*}$ then the simulation is equivalent to the Game 3; otherwise, it equals this game. Due to the security of the PKE scheme, the advantage of $A$ in distinguishing between this game and the previous game is bound by $\epsilon_{PKE}$. Thus we have that

$$\Pr[S_4] \leq \Pr[S_5] + \ell \cdot \epsilon_{PKE}.$$
We could obtain the overall advantage of adversary showed in this lemma via collecting the probabilities from Game 0 to Game 5.

\[ \square \]

**Lemma 5.** Assume that the KE protocol is \((t, \epsilon_{KE})\)-passively secure, the PKE scheme is \((q_{pke}, t, \epsilon_{PKE})\)-secure against adaptive chosen-ciphertext attacks, and the hash function CRHF is \((t, \epsilon_{CRHF})\)-secure and the MAC scheme is \((1, t, \epsilon_{MAC})\)-secure against forgeries under adaptive chosen-message attacks. Then for any adversary running in time \(t\), the probability of \(A\) to correctly answer the Test-query is at most \(1/2 + \epsilon_{ind}\) with

\[ \epsilon_{ind} \leq \epsilon_{CRHF} + \rho \ell \cdot (\ell \cdot \epsilon_{PKE} + \epsilon_{MAC} + \rho \ell \cdot \epsilon_{KE} + \epsilon_{KE}), \]

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

**Proof.** Let \(S_2^\delta\) denotes the event that the \(A\) correctly guesses the bit \(b\) sampled by the Test-query in Game \(\delta\). Let \(\text{Adv}_\delta := \Pr[S_2^\delta] - 1/2\) denote the advantage of \(A\) in Game \(\delta\). Let oracle \(\pi_{t_*}^i\) denote the test oracle which should be mCK-fresh and \(\pi_{t_*}^j\) denote the oracle having matching session to \(\pi_{t_*}^i\).

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

\[ \Pr[S_0^2] = \epsilon_{ind} + 1/2 = \text{Adv}_0 + 1/2. \]

**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

\[ \text{Adv}_0 \leq \text{Adv}_1 + \epsilon_{auth} \leq \text{Adv}_1 + \rho \ell \cdot \epsilon_{KE} + \epsilon_{CRHF} + \rho \ell \cdot (\ell \cdot \epsilon_{PKE} + \epsilon_{MAC}). \]

**Game 2.** This game is similar to the previous game. However, the challenger \(C\) now guesses the partner oracle \(\pi_{t_*}^j\) that stays fresh and participates with \(\pi_{t_*}^i\) in the test session. \(C\) aborts if its guess is not correct. Thus we have that

\[ \text{Adv}_1 \leq (\rho \ell)^2 \cdot \text{Adv}_2. \]

We are now in a game where both oracles accept and the adversary cannot make active attacks.
4 New Modular Compilers for Authenticated Key Exchange

Game 3. Finally, we replace the key $k^*$ of the test oracle $\pi^*_{i}$ and its partner oracle $\pi^*_{j}$ with the same random value $\tilde{k}^*$. Note that the KE protocol instance executed between the test oracle and its partner oracle only can only be attacked by a passive adversary due to previous games. Applying the security of key exchange protocol we obtain that

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

In this game, the response to the Test query consists always of an uniformly random key, which is independent of the bit $b$ sampled in the Test query. Thus we have $\text{Adv}_3 = 0$.

We obtained the advantage of adversary showed in this Lemma by putting together all probabilities from Game 0 to Game 3.

\[\square\]

4.4 Compiler for AKE Protocol From KEM and MAC

In this section we present a generic compiler to construct AKE protocol without random oracles which uses a KEM and a MAC scheme. We first introduce the security primitives used in our construction.

4.4.1 Protocol Description

Let us now describe our KEM-MAC based AKE compiler. Again we assume that the compiled protocol is a Case 1 or Case 2 protocol as in Section 4.2.

The compiler takes as input the following building blocks:

- A passively secure key exchange protocol $\text{KE}$ without long-term key,
- a key encapsulation mechanism scheme $\text{KEM} = (\text{KEM.KGen}, \text{KEM.EnCap}, \text{KEM.DeCap})$, and
- a regular message authentication code scheme $\text{MAC} = (\text{MAC.KGen}, \text{MAC.Tag})$.

To initiate the public system parameter $pms := (pms^{ke}, pms^{kem})$, one may run $pms^{ke} \leftarrow \text{KE.Setup}(1^n)$ and $pms^{kem} \leftarrow \text{KEM.Setup}(1^n)$. The long-term keys of a party $\hat{A}$ are generated as $(sk_{\hat{A}}, pk_{\hat{A}}) \leftarrow \text{KEM.KGen}(pms^{kem})$.

Protocol Execution: The compiled protocol between two parties $\hat{A}$ and $\hat{B}$ proceeds as follows, which is also depicted in Figure 4.4.

1. $\hat{A}$ and $\hat{B}$ run the key exchange protocol. Throughout this protocol run, both parties compute the key $k$ and record the transcript as $T_{\text{KE}}^{\hat{A}}$ and $T_{\text{KE}}^{\hat{B}}$, where $T_{\text{KE}}^{\hat{D}}$ consists of the list of all messages sent and received by party $\hat{D} \in \{\hat{A}, \hat{B}\}$.

Each party $\hat{D} \in \{\hat{A}, \hat{B}\}$ sets the session identifier as $\text{sid} = T_{\text{KE}}^{\hat{D}}$.  

61
Upon receiving ciphertext $\hat{B}$, $\hat{A}$ decapsulates the key $K_{\hat{A}}$ and computes a MAC $M_{\hat{A}} := \text{MAC.Tag}(K_{\hat{A}}, "1" || T_{1}^{\hat{A}})$ over transcript $\text{"1" || T_{1}^{\hat{B}}}$ where $T_{1}^{\hat{B}} := T_{\text{KE}}^{\hat{B}} || \hat{A} || C_{\hat{B}} || \hat{B}$. $\hat{B}$ sends messages ($C_{\hat{B}}, M_{\hat{B}}$) to $\hat{A}$.

Upon receiving messages ($C_{\hat{B}}, M_{\hat{A}}$), $\hat{A}$ rejects if $M_{\hat{B}} \neq \text{MAC.Tag}(K_{\hat{B}}, "2" || T_{1}^{\hat{B}})$. Next, $\hat{A}$ decapsulates the key $K_{\hat{B}} := \text{KEM.DeCap}(sk_{\hat{B}}, C_{\hat{B}})$ and computes $M_{\hat{A}} := \text{MAC.Tag}(K_{\hat{A}}, "1" || T_{1}^{\hat{B}})$. To this end, $\hat{A}$ sends $M_{\hat{A}}$ to $\hat{B}$ and accepts.

Upon receiving message $M_{\hat{B}}$, $\hat{B}$ accepts if $M_{\hat{B}} = \text{MAC.Tag}(K_{\hat{B}}, "1" || T_{1}^{\hat{B}})$.

**Implementation and Session States:** The implementation of this protocol is similar to the PKE-based compiler in Section 4.3. For this protocol, we specify the KEM.DeCap and relative checks using decapsulation key need to be done secure device.

Thus we assume the ephemeral secret vector $esk$ used in each KE protocol instance and the random key $K_{\hat{A}}$ and $K_{\hat{B}}$ used by KEM.EnCap (nor the key decrypted by
KEM.DeCap), will be stored in the variable \textit{st}.

### 4.4.2 Security Analysis

We get the following security result in the mCK model.

**Theorem 3.** Assume that the KE protocol without long-term key is \((t, \epsilon_{\text{KE}})\)-passively secure, the KEM scheme is \((q_{\text{kem}}, t, \epsilon_{\text{KEM}})\)-secure against adaptive chosen-ciphertext attacks and the message authentication code scheme MAC is \((q_{\text{mac}}, t, \epsilon_{\text{MAC}})\)-secure against forgeries under adaptive chosen-message attacks. Then the above protocol is a \((t', \epsilon)\)-mCK-secure AKE protocol in the sense of Definition 25 with \(t' \approx t\), \(q_{\text{kem}} \geq \rho\) and \(q_{\text{mac}} \geq \rho\), and holds that

\[
\epsilon \leq \rho \ell \cdot (2\ell \cdot \epsilon_{\text{KEM}} + 2 \cdot \epsilon_{\text{MAC}} + (\rho \ell + 2) \cdot \epsilon_{\text{KE}}).
\]

The security proof of the above compiler follows the same outline as that of the PKE-based compiler. The main difference is that an adversary can replay the encapsulated keys for all sessions of the party that holds the decapsulation key. Essentially this is because the ciphertext is not bound to a concrete session. However, encapsulated random keys generated by the KEM can be more efficient than encrypted key/hash pairs when using PKE. Technically this alternative approach requires to use a MAC in the last step of the protocol where the adversary is allowed to make up to \(\rho \ell\) queries to a tagging oracle (to be able to simulate successfully). This typically requires the regular MAC to be secure under some complexity assumption whereas in our second compiler we can use a secure one-time MAC with weaker assumption.

**Lemma 6.** If the KE protocol is \((t, \epsilon_{\text{KE}})\)-passively secure, the KEM scheme is \((q_{\text{kem}}, t, \epsilon_{\text{KEM}})\)-secure against adaptive chosen-ciphertext attacks and the MAC scheme is \((q_{\text{mac}}, t, \epsilon_{\text{MAC}})\)-secure against forgeries under adaptive chosen-message attacks, then the above protocol meets the security property 1.) of mCK security definition except for probability with

\[
\epsilon_{\text{auth}} \leq \rho \ell \cdot (\epsilon_{\text{KE}} + \ell \cdot \epsilon_{\text{KEM}} + \epsilon_{\text{MAC}}),
\]

where all quantities are as the same as stated in the Theorem 3.

**Proof.** Let \(S_0^1\) be the event that (i) there exists a mCK-fresh oracle \(\pi_i^s\) that reaches the internal state \(\Phi_i^s = \text{accept}\) with intended communication partner ID\(j\), but (ii) there is no oracle \(\pi_j^t\) such that \(\pi_i^s\) and \(\pi_j^t\) have matching sessions, in Game \(\delta\). We consider the following sequence of games.

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

\[
\Pr[S_0^1] = \epsilon_{\text{auth}}.
\]
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 $\text{abort}_{\text{eph}}$ and aborts, if two oracles output the same ephemeral keys. Thus, the event $\text{abort}_{\text{eph}}$ occurs with probability $\Pr[\text{abort}_{\text{eph}}] \leq \rho \ell \cdot \epsilon_{\text{KE}}$ due to the Lemma 1. We have that

$$\Pr[S_0^1] \leq \Pr[S_1^1] + \rho \ell \cdot \epsilon_{\text{KE}}.$$ 

Game 2. This game proceeds as the previous game, but now the challenger aborts if it fails to guess the first mCK-fresh oracle $\pi^+_{s^*i}$ that accepts without matching session. Since there are $\ell$ parties and $\rho$ oracles for each party, then the probability that the challenger guesses correctly is at least $1/(\rho \ell)$. Thus we have that

$$\Pr[S_1^2] \leq \rho \ell \cdot \Pr[S_2^2].$$

In the sequel, we assume that the guess by challenger is always correct.

Game 3. This game proceeds as exactly as before, but we replace the key $K^*_i$ with random key $K^*_i$ to verify the received MAC $M^*_j$. We do the same modification on the oracles $\{\pi^+_j\}$ which receives the ciphertext $C^*_i$ generated by oracle $\pi^*_i$, and use $K^*_i$ to compute the MACs for oracles $\{\pi^+_j\}$. Since $ID_j$ is uncorrupted. Any adversary $A$ distinguishes this game from previous game argue that the breaking of the KEM scheme. Specifically, we could construct an efficient algorithm $B$ to break the KEM scheme (which is given access to a decryption oracle $\text{DEC}$) by running $A$ as subroutine. When given a under attacked public key $pk^*$, $B$ simulates the AKE game for $A$. $B$ firstly guesses an index $j \overset{\$}{\leftarrow} [\ell]$ pointing to the intended communication partner $ID_j$ of the under attacked oracle $\pi^*_i$, and sets $pk_{ID_j} = pk^*$. Next $B$ sets $pk_{ID_j} = pk^*$ and choosing all other long-term public/private keys honestly. Meanwhile, $B$ computes the ciphertext $C^*_i$ by query challenge ciphertext from KEM security experiment. As for those oracles which receives $C^*_i$, $B$ uses $K^*_i$ obtained from KEM experiment to compute outgoing MACs if necessary; and for other oracles of $ID_j$, $B$ queries $\text{DEC}(sk^*, \cdot)$ oracle to obtain the MAC keys and to compute MACs. Note that if the $K^*_i$ is true key and $B$ guesses correctly with probability $1/\ell$ then the simulation is equivalent to the Game 2; otherwise, it equals this game. Due to the security of KEM scheme, the advantage of $A$ distinguishing between this game and previous game is bound by $\epsilon_{\text{KEM}}$. Thus we have that

$$\Pr[S_2^3] \leq \Pr[S_3^3] + \ell \cdot \epsilon_{\text{KE}}.$$
Game 4. This game proceeds as exactly as before, but the challenger raises an event abort_mac and aborts if the following condition holds: for the fresh oracle π_s^* such that π_s^* ‘accepts’,

- π_s^* has transcript T_{1,1}^{s,s'} = T_{KE}^{s,s'} || ID_i || C_{s}^{s'} || ID_j || C_{j}^{s'}.
- there is no unique oracle π_t^* which has transcript T_{1,1}^{t,t} = T_{KE}^{t,t} || ID_i || C_{t}^{t} || ID_j || C_{j}^{t} satisfying T_{1,1}^{t,s} = T_{1,1}^{t,t}.
- but the MAC value M_s^* received by π_s^* and computed over ‘1’||T_{1,1}^{s,s'} (resp. ‘2’||T_{1,1}^{s,s'}) verifies correctly under the key K^{s,s'}.

If the event abort_mac happens with non-negligible probability, then we could construct a MAC forgery F by running the A as a subroutine and exploiting the forged MAC value by A (as listed above). This is possible, since the MAC K^{s,s'} key used by the oracle π_s^* is a random value due to the previous game. F can compute and verify the MACs involving K^{s,s'} by querying the MAC.Tag(K^{s,s'}, ·) oracle that is given to F. Thus we have that

\[ \Pr[S_3] \leq \Pr[S_4] + \epsilon_{MAC}. \]

Note that in this game the mCK-fresh oracle π_s^* accepts only if there exists an unique π_t^* such that π_s^* and π_t^* having matching sessions, as the game is aborted otherwise. Thus, no adversary can break the authentication property of considered protocol in Game 4, i.e. \( \Pr[S_3] = 0 \). Sum up probabilities from Game 0 to Game 4, we obtained the result of this Lemma. \( \square \)

Lemma 7. If the KE protocol is \((t, \epsilon_{KE})\)-passively secure, the KEM scheme is \((q_{kem}, t, \epsilon_{KEM})\)-secure against adaptive chosen-ciphertext attacks and the MAC scheme is \((q_{mac}, t, \epsilon_{MAC})\)-secure against forgeries under adaptive chosen-message attacks, then for any adversary running in time \( t \), the probability of \( A \) to correctly answer the Test-query is at most \( 1/2 + \epsilon_{ind} \) with

\[ \epsilon_{ind} \leq \rho \ell \cdot (\ell \cdot \epsilon_{KEM} + \epsilon_{MAC} + (\rho \ell + 1) \cdot \epsilon_{KE}), \]

where all quantities are as the same as stated in the Theorem 3.

Proof. Let \( S_3^\delta \) denotes the event that the \( A \) correctly guesses the bit \( b \) sampled by the Test-query in Game \( \delta \). Let \( \text{Adv}_0 := \Pr[S_3^\delta] - 1/2 \) denote the advantage of \( A \) in Game \( \delta \). Let oracle \( \pi_s^* \) denote the test oracle which is fresh and \( \pi_t^* \) denote the oracle having matching session to \( \pi_s^* \).

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

\[ \Pr[S_0^\delta] = \epsilon_{ind} + 1/2 = \text{Adv}_0 + 1/2. \]
4.5 Take Care of the Session Identifier

For the definition of security, the precise definition of what defines a session (the session identifier) is of utmost importance. The queries allow the adversary to ask for the session keys or session states of any oracle that does not belong to the same session via a RevealKey query and a StateReveal query. Moreover the adversary may ask for the long-term keys of parties via a Corrupt query. Depending on the strength of the protocol a Corrupt query may not only be queried for unrelated parties but also for parties involved (via one of their oracles) in the session that is attacked by the adversary, for example to model KCI attacks or forward secrecy. It is not hard to see that the mCK security definition covers KCI resilience property since we allow the adversary to corrupt the test oracle. However there is a non-trivial problem on the selection of session identifier for our generic compilers, when we model KCI attacks and leakage of secret states. In a nutshell if we choose the transcript of all messages sent/received of a session as session identifier then the signature-based compiler is insecure then. One could think of for example replacing the matching sessions of mCK model with the matching conversations [BR94]. We first recall the refined definition about matching conversations in [JKSS12]. Let $T_i^s$ denote the transcript of messages sent and received by an oracle $\pi_i^s$, and $|T_i^s|$ 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 $\omega := |T_i^s|$ and $\nu := |T_j^t|$. We say
that $T_i^s$ is a prefix of $T_j^d$ if $0 < \omega \leq \nu$ and the first $\omega$ messages in transcripts $T_i^s$ and $T_j^d$ are pairwise equivalent as binary strings.

**Definition 30 (Matching conversations (MC) [JKSS12]).** We say that $\pi_i^s$ has a *matching conversation* to oracle $\pi_j^d$, if

- $\pi_i^s$ has sent all protocol messages and $T_j^d$ is a prefix of $T_i^s$, or
- $\pi_j^d$ has sent all protocol messages and $T_i^s = T_j^d$.

To elaborate this problem, we have the following attack on signature-based compiler with the assumptions: (i) partnership is defined via $s\text{id}$ which may include transcript of all messages exchanged, e.g. via matching conversations, (ii) the randomized signature scheme is used. Then attacker $\mathcal{M}$ can perform the attack as:

1. Corrupt the test oracle $\pi_i^s$ (with intended communication partner $\text{ID}_j$) and intercept the signature $\sigma_i^s$ from $\pi_i^s$ which is computed on transcript $T_1 := T_{KE}|\text{ID}_i||\text{ID}_j$.

2. Compute another signature $\sigma_M$ on $T_1$ using the $\text{ID}_i$’s long-term private key, which is different from $\sigma_i^s$. This is possible since the signature scheme we assumed is randomized.

3. Send $\sigma_M$ to the oracle $\pi_j^d$ which shares the same $T_1$ as $\pi_i^s$. Note that $\pi_j^d$ would accept, since the signature $\sigma_M$ is also correctly formed on $T_1$.

4. Reveal the session key $k_j^d$ of $\pi_j^d$ to answer the test query. Note that the key $k_j^d$ that is equivalent to the key $k_i^s$ of $\pi_i^s$. The attacker can reveal the session key from $\pi_j^d$, since it has no matching conversation to $\pi_j^d$ – namely the $\sigma_M$ is not from $\pi_i^s$.

Thus the above attack holds under the assumed assumptions.

However, this attack is not practical. More problematically, there is no simple way to reduce this adversary to break the EUF-CMA security of the signature scheme as Definition 5 that is required by our signature-based compiler. Corresponding to this notion, an adversary is only successful if it manages to produce a signature on a new message. We further highlight that similar problem might be caused also by StateReveal query, e.g. the exposed states are enough to enable adversary to compute the key of randomized message authentication code (MAC) scheme. Therefore one should be very carefully to define the session identifier to specific protocol. In the sequel, we just call the attack based on the similar attack idea as ‘Randomized authentication primitive’ (RAP) attack, and we generalize the conditions of RAP attack as following:

- C1: The considered AKE protocol $\Sigma$ is constructed involving randomized authentication primitive.
• C2: The security of $\Sigma$ is proved in a model $\mathcal{M}$ where the partnership is defined involving the message generated by underlying randomized authentication primitive. We call this kind of partnership definition as randomized partnership (RP).

• C3: The test oracle is allowed to be corrupted or StateReveal query is given.

• C4: The restrictions of adversary on StateReveal query (if it is allowed) and RevealKey query are formulated based on RP.

Thus a protocol $\Sigma$ should take care of the RAP attack if it meets the above conditions. We emphasize that the above attacker seemingly does not break the protocol in any practically way, but it may invalid the security proof in corresponding model. However, some previous works unfortunately overlook this attack that may trivially invalid their security result. We informally list the problematic protocols in the Table 4.1, which are not secure in corresponding model due to the RAP attacks. In order to outline how the adversary launches the RAP attacks, we let ‘CTest’ denote the case that adversary corrupts test oracle, ‘RState’ denote the situation that adversary reveals states from the oracle which contributes the ephemeral key received by test oracle, ‘RKey’ denote the situation that adversary reveals session key from the oracle which shares the same key as test oracle but is not partnered to test oracle. Note that the models that these protocols are proven, allow the adversary to issue attacks denoted by RState, RKey and CTest. We let ‘yes’ denote adversary issues corresponding attack and ‘-’ otherwise. We also describe whether or not the partnership is defined including the messages from randomized primitive in the model they proved. Let ‘WT’ denote whole transcript of protocol messages. Let ‘MAC’ denote randomized message authentication code scheme, and ‘SIG’ denote the randomized signature scheme.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>RAP Attack</th>
<th>Primitive</th>
<th>Partnership</th>
<th>CTest</th>
<th>RState</th>
<th>RKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>KAM [JKL06]</td>
<td>yes</td>
<td>MAC</td>
<td>WT</td>
<td>-</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>KAM [JKL06]</td>
<td>yes</td>
<td>MAC</td>
<td>WT</td>
<td>yes</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>Protocol 2 [SEVB10]</td>
<td>yes</td>
<td>SIG</td>
<td>WT</td>
<td>yes</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>Compiler-MAC [BG11]</td>
<td>yes</td>
<td>MAC</td>
<td>WT</td>
<td>yes</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>Compiler-SIG [CF12]</td>
<td>yes</td>
<td>SIG</td>
<td>WT</td>
<td>yes</td>
<td>-</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 4.1: RAP attacks on Protocols

Please note that although the Compiler-SIG [CF12] specifies the signature scheme to be deterministic (in order to be secure against EphemeralKey query), the authors claim (in [CF12, Remark 1]) that the signature scheme could be randomized after removing the EphemeralKey query and related restrictions from the model. However, then the randomized signature based compiler is not secure in such simplified model. Notice also that Compiler-MAC [BG11] doesn’t change the key derivation function (KDF) of compiled protocol, see its concrete example of Protocol 5 [BG11]. The Compiler-MAC is subject to RAP attacks since it claims in [BG11, Theorem 1] that the resultant protocol
of the compiler remains secure in the model wherever original compiled protocol secure in. However the resultant protocol, for instance the Protocol 5 [BG11] which compiles the NAXOS protocol [LLM07], is not secure in the eCK model [LLM07]; because it satisfies all RAP attack conditions: C1, C2, C3 and C4.

**Solutions for Avoiding RAP Attacks.** In the following we propose several solutions to cope with the above attacks. One can either modify the protocols or the security definitions to re-obtain provable security.

1. Deterministic Primitives. The first and probably most simple solution is to only use deterministic primitives in protocols. The advantage of this solution is that it does not necessarily modify the protocol. However, this approach leaves out the open question: under which condition the protocol can use randomized authentication primitive.

2. Modify the Key Derivation Function. With respect to the second solution, the false protocol could add all messages in its KDF when generating session key. However, this approach might be only suitable for a certain class of protocol, e.g. KAM [JKL06], Compiler-MAC [BG11] and compiler-SIG [CF12]. But it is not applicable for our generic compiler or protocols akin to our compiler that key exchange is executed first and then explicit authentication protocol is executed. Since the adversary may issue StateReveal query to reveal secret input to KDF from non-partnered oracle of test oracle.

3. Choose appropriate Model. One can select a model where the output of randomized authentication primitive is excluded while defining partnership. As the result shown by Theorem 1, the RAP attack can be avoided by defining partnership via partial messages exchanged such as our compilers.

The last approach is more general than the first two approaches. First of all, the approach exploiting deterministic authentication primitive must exclude too much randomize schemes, that is unnecessary. On the second, the approach modifying KDF cannot be applied to all AKE protocols when they are proved in certain strong security model. However, the approach excluding the output of randomized authentication primitive in the definition of partnership, can be applied to all protocols which are subject to RAP attacks (in corresponding models wherein those protocols are proved), without restricting the authentication primitive and requiring modifications on those protocols.
5 One-round Two Party Authenticated Key Exchange

In this chapter, we focus on the constructions for eCK-secure one-round 2AKE protocols in the standard model without NAXOS trick [LLM07]. These proposed protocols are proved secure in the eCK model i.e. in the sense of Definition 27. As mentioned before, the eCK model is known to be one of the strongest AKE model for two party case, which covers almost all desirable security properties.

5.1 Background

5.1.1 Notations for One-round Two Party AKE

In a one-round two party AKE protocol (OR2AKE), each party may send a single ‘message’ and this message is always considered to be independent of the messages sent by the other parties without loss of generality. The independence property of sent messages is required since the session participants can’t achieve mutual authentication in one-round and it enables parties to run protocol instances simultaneously (which is a key feature of one-round protocol). The key exchange procedure is done within two pass and a common shared session key is generated to be known only by session participants.

A general PKI-based OR2AKE protocol may consist of four polynomial time algorithms (OR2AKE.Setup, OR2AKE.KGen, OR2AKE.MF, OR2AKE.KF) with following semantics:

- \( pms \leftarrow \text{OR2AKE.Setup}(1^\kappa) \): On input \( 1^\kappa \), outputs \( pms \), a set of system parameters.

- \((sk_{ID}, pk_{ID}, pf_{ID}) \overset{}{\leftarrow} \text{OR2AKE.KGen}(pms, ID)\): This algorithm takes as input system parameters \( pms \) and a party’s identity \( ID \in IDS \), and outputs a pair of long-term private/public key \( (sk_{ID}, pk_{ID}) \in \{\mathcal{PK}, \mathcal{SK}\} \) for party \( ID \) and a non-interactive proof \( pf_{ID} \) for \( pk_{ID} \) (which may be required during key registration).

- \( m_{ID_1} \overset{}{\leftarrow} \text{OR2AKE.MF}(pms, sk_{ID_1}, ID_2, pk_{ID_2}, r_{ID_1}) \): This algorithm takes as input system parameters \( pms \) and the sender \( ID_1 \)’s secret key \( sk_{ID_1} \), the receiver \( ID_2 \)’s public key \( pk_{ID_2} \) and a randomness \( r_{ID_1} \overset{}{\leftarrow} \mathcal{R}_{\text{OR2AKE}} \), and outputs a message \( m_{ID_1} \in \mathcal{M}_{\text{OR2AKE}} \) to be sent, where \( \mathcal{R}_{\text{OR2AKE}} \) is the randomness space and
$M_{\text{OR2AKE}}$ is message space. We remark that the secret key $sk_{ID_1}$ of sender, the identity $ID_2$ and public key $pk_{ID_2}$ of receiver are only optional for generating the message.

- $K \leftarrow \text{OR2AKE.KF}(pms, sk_{ID_1}, ID_2, pk_{ID_2}, r_{ID_1}, m_{ID_2})$: This algorithm take as the input system parameters $pms$ and $ID_1$’s secret key $sk_{ID_1}$, a public key $pk_{ID_2}$ of $ID_2$, a randomness $r_{ID_1} \leftarrow R_{\text{OR2AKE}}$ and a received message $m_{ID_2}$ from party $ID_2$, and outputs session key $K \in K_{\text{OR2AKE}}$, where $K_{\text{OR2AKE}}$ is the session key space of OR2AKE.

We say that the $\text{OR2AKE.KF}$ algorithm is correct, if for all $(sk_{ID_1}, pk_{ID_1}) \leftarrow \text{OR2AKE.KGen}(pms, ID_1)$ and $(sk_{ID_2}, pk_{ID_2}) \leftarrow \text{OR2AKE.KGen}(pms, ID_2)$, for all $r_{ID_1}, r_{ID_2} \leftarrow R_{\text{OR2AKE}}$ and for all messages $m_{ID_1} \leftarrow \text{OR2AKE.MF}(pms, sk_{ID_1}, ID_2, pk_{ID_2}, r_{ID_1})$ and $m_{ID_2} \leftarrow \text{OR2AKE.MF}(pms, sk_{ID_2}, ID_1, pk_{ID_1}, r_{ID_2})$, it holds that

\[
\text{OR2AKE.KF}(pms, sk_{ID_1}, ID_2, pk_{ID_2}, r_{ID_1}, m_{ID_2}) = \text{OR2AKE.KF}(pms, sk_{ID_2}, ID_1, pk_{ID_1}, r_{ID_2}, m_{ID_1})
\]

The Figure 5.1 briefly illustrates the generic protocol execution of OR2AKE protocol.

![Figure 5.1: General OR2AKE Protocol](image)

**NAXOS Trick Revisit.** One of the most prominent open questions in the research field on AKE is how to securely implement the NAXOS trick [LLM07]. NAXOS trick is a technique that is introduced to hide the exponent (or de-facto ephemeral key) of an ephemeral public key from an adversary even if the adversary obtains the ephemeral secret key. Please first note that there might exist some variants of NAXOS trick in which the prominent example is the twisted-PRF trick recently used in FSXY scheme [FSXY12]. In the sequel, we are going to call all those variants as NAXOS trick.

---

1. The twisted-PRF trick is first used in Okamoto’s construction [Oka07] to satisfy the eCK security in the standard model.
In a typical Diffie-Hellman key exchange protocol, the ephemeral secret key $\tilde{x}$ is used as the exponent of the ephemeral public key as $X = g^{\tilde{x}}$. However, in a protocol designed with NAXOS trick, taking the twisted-PRF trick as example, the exponent of the ephemeral public key is generated as $x := \text{PRF}(\tilde{x}', a) \oplus \text{PRF}(a, \tilde{x})$. Therefore, even though the security model allows an adversary to obtain ephemeral secret key $\tilde{x}$, but the exponent of the ephemeral public key is still not exposed to the adversary. However, after applying the twisted-PRF trick, the attacker’s interest may still be the exponent of Diffie-Hellman key. Then how to protect such exponent remains an open problem.

Exploiting secure device might be one natural solution to protect the output of such trick, since the twisted-PRF trick relies on long-term key which is always stored on secure device and should never be directly passed to host machine. Then such trick can be realized as follows: (i) the host machine generates the ephemeral key $\tilde{x}$ and passes it to secure device; (ii) and then the secure device computes the de-facto ephemeral private key $x := \text{PRF}(\tilde{x}, a) \oplus \text{PRF}(a, \tilde{x})$ and keeps the $x$ securely as long-term key, (iii) then the secure device computes the ephemeral public key $X = g^x$ and sends it back to host machine. If the secure device sends the new generated ephemeral private $x$ back to host machine where it will be used then it is also vulnerable to attacks aiming to the original $\tilde{x}$ chosen at host machine.

Eventually, we could conclude that the NAXOS trick might be necessary to achieve eCK security if and only if the secure device is unable to generate the randomness. Otherwise the NAXOS trick can be seen identically to choosing the exponent $x$ from a leakage-free random source, e.g. randomness generator of secure device. Besides, all other computations related to NAXOS trick might need to be done on secure device too, e.g. encapsulation algorithm of KEM in the FSXY protocol [FSXY12] which takes as input the ephemeral key generated by NAXOS trick. Since the ephemeral private key generated by NAXOS trick should be generated only on secure device and never be returned to host machine. Those computations would dramatically increase the burden of secure device. On the other hand, the protocol with NAXOS trick is HS-round inefficient since it might need at least two HS-round, in which one round is used for generate the ephemeral public key and the second round might be used to generate the final session key. In contrast, for a protocol without NAXOS trick, only one HS-round might be enough for session key generation.

5.1.2 Motivations

So far there are only few AKE protocols are provably secure without random oracles in the eCK model. Although the protocols [Oka07, MO11a] are proved to be eCK secure in the standard model, they require a rather strong class of pseudo-random function family with pairwise independent random sources (which is referred to as $\pi$PRF) as key derivation function (KDF). Most recently, Fujioka et al. [FSXY12] introduced a generic
construction for one-round two party AKE from key encapsulation mechanism (KEM) which is generalized from BCNP [BCNP08]. Although the FSXY scheme [FSXY12] is shown to be eCK (CK+) secure in the standard model, it is built relying on a special twisted-PRF trick. The session states defined in FSXY scheme only include the random values used to execute the both twisted-PRF trick and IND-CPA KEM. However it is not hard to see if either the ephemeral private key generated by twisted-PRF trick or the encapsulation key of test session is allowed to be leaked via StateReveal query (as the BCNP scheme), then the FSXY protocol is insecure in the eCK model. As discussed above, to securely implement the FSXY protocol, one might need to distribute all computations related to twisted-PRF trick on secure device. This would lead to inefficiency of implementation of the FSXY protocol with secure device. Another drawback of FSXY protocol is that it might be unable to be executed simultaneously by session participants. Since, in the protocol description of FSXY, the responder cannot generate the outgoing ciphertext of IND-CPA KEM until it received the ephemeral public key sent by initiator. Hence the FSXY may lose an important feature of one-round AKE protocol.

So far, to our best of knowledge it is still an open question to construct eCK secure protocols without random oracles and without NAXOS trick under standard assumptions (e.g. without πPRF). As those secure devices might be short in both storage capacity and computational resource, the algorithm on secure device is often causing performance bottleneck of systems. Therefore it is necessary to seek efficient eCK secure construction which can be efficiently implemented with secure device.

5.1.3 Contributions

We first present an authenticated key exchange protocol (named GC-KKN) in Section 5.2 to solve the above open problem. In the construction of GC-KKN, we exploit the building blocks including strong randomness extractor (SEXT) family, and pseudo-random function (PRF) family, passively secure one-round key exchange (KE) protocols, IND-CCA secure key encapsulation mechanism (KEM) schemes, and CKS-light secure non-interactive key exchange (NIKE) schemes [FHKP13] (that is secure against chosen identity and long-term public key attacks). The NIKE can be instantiated with the schemes based on factorization problem or Diffie-Hellman problem as proposed by Freire et al. [FHKP13]. Similar to FSXY protocol, the KEM in our construction could be instantiated using any IND-CCA secure scheme based on hardness of factorization problem, code-based problem, lattice-based problem and so forth. As opposed to FSXY scheme, GC-KKN does not rely on any NAXOS like trick, that yields a more efficient solution when it is implemented with secure device. We give compact game-based proofs reducing eCK security of GC-KKN to break the used cryptographic primitives without random oracles.
On the second we present a practical AKE protocol (P1) in Section 5.3 that is eCK secure under standard assumptions (e.g. without πPRF). The proposed protocol is based on bilinear pairings, target collision resistant hash function family, and pseudo-random function family. To be of independent interesting, P1 is able to run under post-specified peer setting [CK02] (i.e. without knowing any information of communication peer at session activation), unlike FSXY scheme and our GC-KKN scheme which might be executed under only pre-specified peer setting (i.e. they need the public key of peer to run the KEM). Our construction idea of P1 is inspired by the GC-KKN. We observe that it is possible to merge those computations in KE, KEM and NIKE schemes if they work under the same algebraic groups. In order to be secure against active attackers, each party (including the attacker) is required to construct some kind of ‘tag’ to encode consistency information on its chosen (either long-term or ephemeral) public keys based on specific weak Programmable Hash Function [HJK11]. Those tags are particularly customized to be independent of any information about receivers, which enables P1 to be able to run in the post-specified peer setting. However we have to make use of the pairing to provide a means of consistency checking that (both long-term and ephemeral) public keys coming from the adversary are in some sense of well-formed. Due to those tags, all public keys are mutually independent. Thus an active adversary is not able to lead non-partnered fresh sessions to generate co-related session keys. In particular, any active adversaries have to leave session key related secret information in those tags which can be extracted and exploited by the challenger (during the proof simulation) using corresponding trapdoor secret, e.g. the exponent of the group elements used to computing the tags. We make use of the fact those trapdoor information can’t be trivially obtained by adversary. Intuitively, the tag is what gives us the necessary leverage to deal with the non-trivial eCK security without resorting to random oracles. Although the consistency check of those tags might be relatively expensive, fortunately all pairing (including partial session key material generation) can be done on more powerful host machine rather than on computational resource-limited secure device (which is only used to generate final session key). In order to securely implement P1, only one exponentiation is required on secure device that is the more efficient than any previous eCK secure protocols without random oracles.

5.1.4 Related Works

In 1986, Matsumoto et al. [MTI20] first studied the Diffie-Hellman based key exchange protocols with implicit key authentication that result in a line of research on one-round AKE. Meanwhile, the most famous and efficient one is the MQV protocol by Menezes, Qu, and Vanstone [MQV95, LMQ+03]. On Crypto 2005 conference, Krawczyk [Kra05] presented a hashed variant of MQV called HMQV, which is formally proven secure in the CHHMQV model, relying on random oracles and strong assumptions (i.e. gap Diffie-
One-round Two Party Authenticated Key Exchange

Hellman assumption [OP01] and knowledge of exponent assumption [BP04]). Krawczyk also showed that HMQV protocol offers more security features than MQV, in particular for resistance to KCI attacks, UKS attacks and the leakage of secret states. However Menezes [Men07] pointed out that the chosen identity and long-term public key attacks on HMQV are not formally studied in [Kra05]. Besides, the application of secure module was also suggested by HMQV protocol [Kra05] to prevent the leakage of intermediate secrets. But the detailed implementation approach was not elaborated.

In 2007 LaMacchia et al. [LLM07] proposed an extended Canetti-Krawczyk (eCK) model, in which the adversary is equipped with a **EphemeralKeyReveal** query to access all ephemeral private input required to carry on session key computations which is similar to the **StateReveal** query in the CK model. Since then, many AKE protocols [Ust08, SEVB10, Ust09, MU08, Oka07, PW11, MO11a], are proposed to capture eCK security. However, most of those schemes are only provably secure in the random oracle model. On the other side, those eCK-secure schemes [Oka07, MO11a] without random oracles require the **πPRF** family as key derivation function that is hard to realize in practical.

As for generic two party AKE constructions, e.g. the BCNP [BCNP08] and FSXY [FSXY12] schemes based on IND-CCA secure KEM, the session states that can be revealed are never clearly defined which makes theirs security analysis incomplete. In particular, we notice that the twisted-PRF technique used in the FSXY protocol (also used in [Oka07]) may need to be implemented with secure device to achieve leakage resilience on new generated ephemeral private key by such trick. As the above discussion regarding NAXOS trick, that might increase unnecessary workload on secure device. Basically the FSXY protocol is built in an inefficient manner. Please note that there are a lot of works such as [MO11a, Ust09, KFU09] which are motivated to construct key exchange protocols without the NAXOS tricks. As well this is also one of the motivations of our work.

With respect to the protection of session states involving secure device, two implementation scenarios (which are referred to as SImplM-I and SImplM-II in the following) and corresponding security models have been studied in literatures [SEVB10, YZ11]. Basically, the implementation of the AKE protocol is divided into two parts which are respectively run at host machine (which is subject to attacks on session states) and at secure device. In corresponding security model, the adversary is able to reveal all possible states stored at host machine, but the secure device is treated as a black-box which is resilience of the leakage of intermediate values. However, Yoneyama et al., in their recent work [YZ11], particularly showed errors in the security proofs of SMQV and FHMQV [SEVB10] when they are realized under SImplM-II. They pointed out that the uniqueness of secret exponents interferes with the simulation to the query to reveal the result of intermediate computations. Thereafter they formally clarified the technical limitations of Diffie-Hellman key exchange protocols for achieving provable security in
the seCK model [SEVB10]. Unfortunately, their results showed that there is no scheme has been provably secure in the seCK model with implementation scenario SimplM-II.

5.1.5 Simplify the Security Proof for One-round Two Party AKE in the eCK Model

Roughly speaking, to prove the security of a protocol in the eCK model, we need to examine the probability of adversary in breaking the indistinguishability of session key of the eCK-fresh test oracle, where the eCK-freshness is defined as Definition 26. Besides the restrictions regarding EstablishParty and RevealKey queries which are ‘deterministic’, we particular may obtain different freshness events on whether the test oracle has matching sessions and corresponding freshness cases related to different combinations of StateReveal and Corrupt queries which are ‘flexible’ and determined by adversary’s choice. However, among those different freshness cases, at least one would occur in the security game. In the following, we list all possibilities about the freshness cases relative to StateReveal and Corrupt queries by expanding the eCK-fresh Definition 26. Therefore, we might need to show the security of each OR2AKE protocol under one of the following complementary events and all freshness related cases:

- **Event 1**: There is a oracle \( \pi^t_B \) held by \( \hat{B} \), such that \( \pi^s_A \) and \( \pi^t_B \) have matching sessions, and we have the following ‘freshness’ related disjoint cases:
  - Case 1 (C1): The adversary did not query Corrupt(\( \hat{A} \)) nor Corrupt(\( \hat{B} \)).
  - Case 2 (C2): The adversary did not query StateReveal(\( \pi^s_A \)) nor StateReveal(\( \pi^t_B \)).
  - Case 3 (C3): The adversary did not query StateReveal(\( \pi^s_A \)) nor Corrupt(\( \hat{B} \)).
  - Case 4 (C4): The adversary did not query Corrupt(\( \hat{A} \)) nor StateReveal(\( \pi^t_B \)).

- **Event 2**: There is no oracle \( \pi^t_B \) held by \( \hat{B} \), such that \( \pi^s_A \) and \( \pi^t_B \) have matching sessions, and we have the following disjoint cases:
  - Case 5 (C5): The adversary did not query Corrupt(\( \hat{A} \)) nor Corrupt(\( \hat{B} \)).
  - Case 6 (C6): The adversary did not query StateReveal(\( \pi^s_A \)) nor Corrupt(\( \hat{B} \)).

In order to complete the proof for a OR2AKE protocol, we must provide the security proofs for all above six cases, that might be tiresome. However, due to the Propositions 1 and 2 from [MO11a], the security proofs for Cases C1, C3 and C4 can be reduced to the security proof for Case C5 or C6. Let us first recall those two propositions in the following that will be used in all proofs of proposed OR2AKE protocols.

In the following, we recall those propositions and rewrite it to fit to our freshness cases.
Proposition 1. If adversary $A_1 (t_1, \epsilon_{A_1})$-breaks the eCK security of an OR2AKE protocol $\Sigma$ in case C4, then there exists adversary $A_2$ who can $(t_2, \epsilon_{A_2})$-breaks the eCK security of $\Sigma$ in case C3, such that $t_1 \approx t_2$ and $\epsilon_{A_1} = \epsilon_{A_2}$.

**Proof.** Intuitively, in cases C3 and C4, the test oracle has matching sessions, then the adversary could select either an oracle $\pi_A^*$ or its partner $\pi_B^*$ as test oracle since $\pi_A^*$ and $\pi_B^*$ will compute the same session key. In both cases the adversary reveals the states of an oracle and Corrupt its peer. We show the security reduction as follows. $A_2$ runs $A_1$ as subroutine and responds all the oracle queries except for the test oracle. When $A_1$ issues the test query to $\pi_A^*$, $A_2$ selects the matching partner $\pi_B^*$ as the test session. When $A_2$ receives the real session key or random key, $A_2$ sends it to $A_1$. If $A_1$ outputs a bit, $A_2$ outputs the same bit. Note that $A_2$ can issue StateReveal($\pi_A^*$) since it has matching session to the ‘test oracle’ (i.e. $\pi_B^*$) from the view of $A_2$. So $A_2$ can correctly respond to all the queries issued by $A_1$. Therefore, if $A_1$ breaks the security of the proposed protocol in case C4, $A_2$ wins the game in case C3 with the same advantage. □

Proposition 2. If adversary $A_1 (t_1, \epsilon_{A_1})$-breaks the eCK security of a OR2AKE protocol $\Sigma$ in case C1 (C3), then there exists adversary $A_2$ who can $(t_2, \epsilon_{A_2})$-breaks the eCK security of $\Sigma$ in case C5 (C6), such that $t_1 \approx t_2$ and $\epsilon_{A_1} = \epsilon_{A_2}$.

**Proof.** This proof is similar to the proof of Proposition 1. $A_2$ runs $A_1$ as subroutine and responds all the oracle queries except for the test oracle. Note that $A_1$ issues the Test query to oracle $\pi_A^*$ which has a matching session to oracle $\pi_B^*$ in the view of $A_1$. Note that the oracle $\pi_B^*$ is simulated by $A_2$. So that when $A_2$ receives the real session key or random key, $A_2$ sends it to $A_1$. If $A_1$ outputs a bit, $A_2$ outputs the same bit. $A_2$ can respond to any oracle queries issued by $A_1$ since the restricted oracle queries between and are equivalent. Therefore, $A_2$ breaks the security of the proposed protocol in case C5 (C6) if $A_1$ wins the game in case C1 (C3). □

Therefore, we can only prove the advantage of the adversary on breaking any OR2AKE protocol is negligible in the eCK model under freshness cases C2, C5 and C6.

5.2 A Generic One-round Two Party AKE Protocol from KE, KEM and NIKE

In this section, we present a generic one-round authenticated key exchange protocol from KE, KEM and NIKE (denoted by GC-KKN), that is more suitable to be implemented for providing eCK security than previous works. In our generic construction, the following building blocks are required:
One-round Two Party Authenticated Key Exchange

- Passively secure key exchange scheme \( KE = (KE\text{-Setup}, KE\text{-EKGen}, KE\text{-SKGen}) \) without long-term key. We particularly require the \( KE\text{-EKGen} \) algorithm executed by two session participants are the same type. So that the KE protocol can be executed simultaneously by two parties. For simplicity and readability, we just assume that the input message in of \( KE\text{-EKGen} \) is \( \emptyset \) and the message \( m \) generated by \( KE\text{-EKGen} \) includes only ephemeral public key in the following protocol description.

- IND-CCA secure key encapsulation mechanism scheme \( KEM = (KEM\text{-Setup}, KEM\text{-KGen}, KEM\text{-EnCap}, KEM\text{-DeCap}) \).

- CKS-light secure non-interactive key exchange scheme \( NIKE = (NIKE\text{-Setup}, NIKE\text{-KGen}, ShareKey) \).

- Strong randomness extractor \( SEXT(\cdot, \cdot) : S_{SEXT} \times M_{SEXT} \rightarrow R_{SEXT} \). We assume that the space \( M_{SEXT} \) matches the key spaces of \( KE, KEM \) and \( NIKE \) schemes.

- Pseudo-random function family \( PRF(\cdot, \cdot) : R_{SEXT} \times D_{PRF} \rightarrow K_{AKE} \).

**Design Principle.** The first general attack we consider is the chosen ephemeral key (CEK) attacks which is mainly formulated via Send query in conjunction with other oracle queries that basically enables adversary to send any ephemeral keys of her own choice to sessions of uncorrupted party which he can reveal session keys (without violating the freshness of test oracle) and to obtain non-trivial information about test oracle. As other attacks in the eCK model, we say an adversary successfully launches a CEK attack on considered AKE protocol \( \Sigma \), if it \((t, \epsilon)\)-breaks the eCK security of \( \Sigma \) under any freshness case listed in Section 5.1.5. Meanwhile we particularly need to cope with CEK attacks under the freshness case \( C6 \) listed in Section 5.1.5, in which case the attack may be launched mainly against an uncorrupted party (which is normally the intended communication partner of test oracle and might be only one that is not corrupted in the AKE security experiment). In order to handle CEK attacks under case \( C6 \), we utilize the IND-CCA secure KEM scheme as both FSXY [FSXY12] and BCNP [BCNP08] schemes. Since the IND-CCA secure KEM is known as one of the most efficient and effective solutions which can withstand this kind of attack. The IND-CCA security is necessary to simulate RevealKey queries to oracles of target uncorrupted party (say the intended communication partner of test oracle) without knowing corresponding long-term private key in the proof simulation. Meanwhile, we might ask \( DEC \) oracle provided by KEM experiment for help.

On the second, we have to consider security of test oracle when its session states are leaked. The worst case (which might be very likely to happen) is that test oracle received an ephemeral key which is chosen by adversary, or the session states are disclosed from both test oracle and its partner oracle. In either case, the adversary may know
all ephemeral secrets used to compute the session key of test oracle without knowing the long-term keys of session participants of test oracle. This is also why the FSXY construction needs expensive NAXOS like trick to generate certain intermediate secret that is leakage resilience. To deal with this situation without NAXOS trick, our solution is to use non-interactive key exchange scheme, namely we make use of the long-term shared key of session participants to compute part of session key. This makes sense due to the fact that the long-term keys of session participants are not corrupted. We need the CKS-light security for NIKE scheme since we have to ‘appropriately’ simulate the session keys of uncorrupted oracles in presence of adversary who can register identities and public keys of her own choice (i.e. via EstablishParty query).

While considering the wPFS attack in which long-term keys of both participants might be leaked. Intuitively, we cannot make use of KEM or NIKE schemes to achieve wPFS security property, since the security of both schemes relies on uncompromised long-term keys. Unlike the FSXY [FSXY12] scheme, we do not adopt to IND-CPA secure KEM (which is called as wKEM in FSXY) in the construction since wKEM cannot be executed simultaneously by two sessions. Namely the responder cannot generate the outgoing ciphertext until it received the ephemeral public key sent by initiator. Our solution avoids this circumstance via exploiting passively secure one-round key exchange protocol KE without long-term keys. This is a generalization from the BCNP construction [BCNP08] wherein only Diffie-Hellman key exchange protocol [DH76] is considered. Our construction is able to be instantiated with any other passively secure one-round key exchange protocols.

The application of strong randomness extractor here is to derive uniform random keys from non-uniform sources of randomness. In our construction, we will make use of SEXT to extract a uniform random key from different random sources for underlying key derivation function, i.e. PRF. This guarantees the key of PRF to be a statistically indistinguishable value from a uniform randomly chosen element from the key space of PRF. As the NIKE key, KEM key and KE may come from different spaces which can not be naively used as input key of PRF. In particular those keys may be not random bit-string uniformly distributed in the key space of PRF, even though they satisfy key indistinguishability property in correspond key spaces. Take the Diffie-Hellman KE protocol [DH76] as example, the passively security of this scheme relies on the Decisional Diffie-Hellman assumption (DDH) [Bon98]. Let $G = \langle g \rangle$ be a cyclic group generated by $g$ of prime order $q$ be a group. The DDH problem states that: given tuple $(g, g^a, g^b, g^{ab}, g^c) \in \mathbb{G}^5$, there is no efficient algorithm that can decide whether $c = ab$, where $a, b$ and $c$ are chosen at random in $[1, \ldots, q]$. The session key of such KE protocol is only a random element of $\mathbb{G}$. However, a perfectly random element of $\mathbb{G}$ with high entropy may be not a perfectly random bit string over other space. Therefore, we have to find a way (i.e. exploiting SEXT here) to generate almost uniform random keys for the session key derivation function from different random sources.
5 One-round Two Party Authenticated Key Exchange

5.2.1 Protocol Description

Set-up: To initiate the system, the public system parameters $pms := (pms^{ke}, pms^{kem}, pms^{nike}, k_{SEXT})$ are firstly generated via performing $pms^{ke} \leftarrow \text{KE.Setup}(1^k)$, $pms^{kem} \leftarrow \text{KEM.Setup}(1^k)$, $pms^{nike} \leftarrow \text{NIKE.Setup}(1^k)$ and $k_{SEXT} \leftarrow \mathcal{S}_{SEXT}$.

Long-term key Generation and Registration: A party $\hat{A}$ may run algorithms $(pk_{\hat{A}}^{kem}, sk_{\hat{A}}^{kem}) \leftarrow \text{KEM.KGen}(pms^{kem})$ and $(pk_{\hat{A}}^{nike}, sk_{\hat{A}}^{nike}, pf_{\hat{A}}) \leftarrow \text{NIKE.KGen}(pms^{nike}, \hat{A})$ to generate the long-term key pair for KEM and NIKE schemes respectively. The public key $pk_{\hat{A}}^{kem}$ can be registered arbitrarily. As well the public key $pk_{\hat{A}}^{nike}$ can be registered arbitrarily if $pf = \emptyset$. Otherwise the $pk_{\hat{A}}^{nike}$ might be registered if the non-interactive proof $pf_{\hat{A}}$ is evaluated to be sound based on $pk_{\hat{A}}^{nike}$. As for certain NIKE scheme, the interactive zero-knowledge proof scheme might be required to ensure the registered public key is consistent (e.g. the factoring based NIKE in [FHKP13]).

\[
\begin{align*}
\hat{A} & \quad sk_{\hat{A}} := (sk_{\hat{A}}^{kem}, sk_{\hat{A}}^{nike}) \\
\hat{A} & \quad pk_{\hat{A}} := (pk_{\hat{A}}^{kem}, pk_{\hat{A}}^{nike}) \\
& \quad \langle esk_{\hat{A}}, epk_{\hat{A}} \rangle \xleftarrow{\$} \text{KE.EKGen}(pms^{ke}, \emptyset) \\
& \quad (K_{\hat{A}}, C_{\hat{A}}) \leftarrow \text{KEM.EnCap}(pk_{\hat{A}}^{kem}) \\
\end{align*}
\]

\[
\begin{align*}
\hat{B} & \quad sk_{\hat{B}} := (sk_{\hat{B}}^{kem}, sk_{\hat{B}}^{nike}) \\
\hat{B} & \quad pk_{\hat{B}} := (pk_{\hat{B}}^{kem}, pk_{\hat{B}}^{nike}) \\
& \quad \langle esk_{\hat{B}}, epk_{\hat{B}} \rangle \xleftarrow{\$} \text{KE.EKGen}(pms^{ke}, \emptyset) \\
& \quad (K_{\hat{B}}, C_{\hat{B}}) \leftarrow \text{KEM.EnCap}(pk_{\hat{B}}^{kem}) \\
\end{align*}
\]

\[
\begin{align*}
& \quad \text{sid} := \hat{A}||\hat{B}||epk_{\hat{A}}||epk_{\hat{B}}||C_{\hat{A}}|C_{\hat{B}} \\
& \quad eK := \text{KE.SKGen}(esk_{\hat{A}}, epk_{\hat{A}}||epk_{\hat{B}}) \\
& \quad K_{\hat{B}} := \text{KEM.DeCap}(sk_{\hat{B}}^{kem}, C_{\hat{B}}) \\
& \quad ShK_{\hat{A}}K_{\hat{B}} := \text{ShareKey}(A, sk_{\hat{A}}^{nike}, B, pk_{\hat{B}}^{nike}) \\
& \quad eK' := \text{SEXT}(eK) \\
& \quad K'_{\hat{A}} := \text{SEXT}(K_{\hat{A}}) \\
& \quad K'_{\hat{B}} := \text{SEXT}(K_{\hat{B}}) \\
& \quad ShK'_{\hat{A}}K'_{\hat{B}} := \text{SEXT}(ShK_{\hat{A}}K_{\hat{B}}) \\
& \quad eK'' := \text{PRF}(eK', \text{sid}) \\
& \quad K''_{\hat{A}} := \text{PRF}(K'_{\hat{A}}, \text{sid}) \\
& \quad K''_{\hat{B}} := \text{PRF}(K'_{\hat{B}}, \text{sid}) \\
& \quad ShK''_{\hat{A}}K''_{\hat{B}} := \text{PRF}(ShK'_{\hat{A}}K'_{\hat{B}}, \text{sid}) \\
& \quad k_{e} := eK'' \oplus K''_{\hat{A}} \oplus K''_{\hat{B}} \oplus ShK''_{\hat{A}}K''_{\hat{B}} \\
\end{align*}
\]

\[
\begin{align*}
& \quad \text{sid} := \hat{A}||\hat{B}||epk_{\hat{A}}||C_{\hat{A}}|epk_{\hat{B}}||C_{\hat{B}} \\
& \quad eK := \text{KE.SKGen}(esk_{\hat{B}}, epk_{\hat{B}}||epk_{\hat{A}}) \\
& \quad K_{\hat{A}} := \text{KEM.DeCap}(sk_{\hat{A}}^{kem}, C_{\hat{A}}) \\
& \quad ShK_{\hat{A}}K_{\hat{B}} := \text{ShareKey}(B, sk_{\hat{B}}^{nike}, \hat{A}, pk_{\hat{A}}^{nike}) \\
& \quad eK' := \text{SEXT}(eK) \\
& \quad K'_{\hat{B}} := \text{SEXT}(K_{\hat{B}}) \\
& \quad K'_{\hat{A}} := \text{SEXT}(K_{\hat{A}}) \\
& \quad ShK'_{\hat{A}}K'_{\hat{B}} := \text{SEXT}(ShK_{\hat{A}}K_{\hat{B}}) \\
& \quad eK'' := \text{PRF}(eK', \text{sid}) \\
& \quad K''_{\hat{B}} := \text{PRF}(K'_{\hat{B}}, \text{sid}) \\
& \quad K''_{\hat{A}} := \text{PRF}(K'_{\hat{A}}, \text{sid}) \\
& \quad ShK''_{\hat{A}}K''_{\hat{B}} := \text{PRF}(ShK'_{\hat{A}}K'_{\hat{B}}, \text{sid}) \\
& \quad k_{e} := eK'' \oplus K''_{\hat{A}} \oplus K''_{\hat{B}} \oplus ShK''_{\hat{A}}K''_{\hat{B}} \\
\end{align*}
\]

\[
\begin{align*}
& \quad \hat{A}, epk_{\hat{A}}, C_{\hat{A}} \xrightarrow{\$} \hat{B}, epk_{\hat{B}}, C_{\hat{B}} \\
\end{align*}
\]

Figure 5.2: Generic One-round AKE Protocol from KE, KEM and NIKE

Protocol Execution: On input $pms := (pms^{ke}, pms^{kem}, pms^{nike}, k_{SEXT})$, the protocol between party $\hat{A}$ and party $\hat{B}$ is proceeded as following which is also informally depicted
in Figure 5.2:

1. Upon activation a session at $A$, it performs the steps: (a) choose ephemeral public/private keys $(eK_A, eK_B) \leftarrow K.EKGen(pms_{ke}, \emptyset)$; (b) run $(K_A, C_A) \leftarrow KEM.EnCap(pk_{B}^{ke})$ and compute $K'_A := SEXT(K_A)$; (c) send $(A, eK_A, C_A)$ to $B$.

2. Upon activation a session at $B$, it performs the steps: (a) choose ephemeral public/private keys $(eK_B, eK_B) \leftarrow K.EKGen(pms_{ke}, \emptyset)$; (b) run $(K_B, C_B) \leftarrow KEM.EnCap(pk_{B}^{ke})$ and compute $K'_B := SEXT(K_B)$; (c) send $(B, eK_B, C_B)$ to $A$.

3. Upon receiving $(A, eK_A, C_A)$, party $B$ does the following: (a) set session identifier $\text{id} := A||B||eK_A||eK_B||C_A||C_B$ where the messages are ordered by round, and within each round lexicographically by the identities of the purported senders; (b) compute $eK := K.ESKGen(eK_B, eK_A||eK_B)$ and $eK'' := SEXT(eK)$ and $eK'' := PRF(eK', \text{id})$; (c) compute $K'' := PRF(K'_B, \text{id})$; (d) compute $(K_A := KEM.DeCap(sk_{A, B}^{ke}, C_A), K'_A := SEXT(K_A'), K'' := PRF(K'_A, \text{id}), (e) compute ShK_{A, B} := ShareKey(A, sk_{A, B}^{nike}, B, pk_{B}^{nike}), ShK_{A, B}^{A, B} := SEXT(ShK_{A, B}^{A, B})$ and $ShK_{A, B}^{A, B} := PRF(ShK_{A, B}^{A, B}, \text{id})$; (f) compute the final session key as $k_e := eK'' \oplus K'' \oplus K'' \oplus ShK''$.

4. Upon receiving $(B, eK_B, C_B)$, $A$ does the following: (a) set session identifier as $\text{id} := A||B||eK_A||eK_B||C_A||C_B$ where the messages are ordered by round, and within each round lexicographically by the identities of the purported senders; (b) compute $eK := K.ESKGen(eK_A, eK_A||eK_B)$ and $eK'' := SEXT(eK), eK'' := PRF(eK', \text{id})$; (c) compute $K'' := PRF(K'_A, \text{id})$; (d) compute $(K_B := KEM.DeCap(sk_{A, B}^{ke}, C_B), K'_B := SEXT(K_B'), K'' := PRF(K'_B, \text{id}), (e) compute ShK_{A, B} := ShareKey(B, sh_{B, A}^{nike}, B, pk_{B}^{nike}), ShK_{A, B}^{A, B} := SEXT(ShK_{A, B}^{A, B})$ and $ShK_{A, B}^{A, B} := PRF(ShK_{A, B}^{A, B}, \text{id})$; (f) compute the final session key as $k_e := eK'' \oplus K'' \oplus ShK''$.

Implementation and Session States: We now define the session states in terms of implementation model with secure device. Basically, all states of K.EKGen, K.ESKGen and KEM.EnCap algorithms would be stored in the state variable $st$. For instance the state of an oracle $\pi_A$ might include values $(eK_A, eK_A', eK''_A, K_A, K_A', K''_A)$ appeared in the above protocol description and other randomness or intermediate values generated within algorithms: K.EKGen, K.ESKGen and KEM.EnCap. Namely those algorithms can be executed on host machine. However, we assume no secret states related to KEM.DeCap and ShareKey algorithms can be revealed. This can be realized by doing all computations involving long-term private key of KEM.DeCap and ShareKey algorithms, and final session key generation on secure device, i.e. processing the steps 3.(d,e,f) and
4.(d,e,f) on secure device. Meanwhile the KEM.DeCap and ShareKey algorithms might involve expensive consistency check operations, say pairing based NIKE. But we stress that those computations of KEM.DeCap and ShareKey algorithms without using long-term private key and without using values generated by long-term private key, could be done on host machine for efficiency consideration.

5.2.2 Security Analysis

We assume without loss of generality that the maximum probability for the event that two oracles output the same ciphertext $C$ or ephemeral public key $epk$, is a negligible fraction $1/2^\lambda$ where $\lambda \in \mathbb{N}$ is a large enough integer in terms of the security parameter $\kappa$. Let $\text{MAX}(X_1, X_2, X_3)$ denote the function to obtain the maximum value from variables $X_1$, $X_2$ and $X_3$.

**Theorem 4.** Suppose that the SEXT is $(\kappa, \epsilon_{\text{SEXT}})$-strong randomness extractor, the KEM is $(q_{\text{kem}}, t, \epsilon_{\text{KEM}})$-secure against adaptive chosen-ciphertext attacks and KE without long-term keys is $(t, \epsilon_{\text{KE}})$-passively secure, and the pseudo-random function PRF is $(q_{\text{prf}}, t, \epsilon_{\text{PRF}})$-secure, and the NIKE is $(t, \epsilon_{\text{NIKE}})$-CKS-light-secure non-interactive scheme. And we assume that either KE key or KEM key or NIKE key has $\kappa$-min-entropy.\(^2\) Then the proposed protocol is $(t', \epsilon)$-eCK-secure in the sense of Definition 27 with $t' \leq t$, $q_{\text{kem}} \geq \rho$, $q_{\text{prf}} \geq \rho + 1$ and

$$\epsilon \leq \frac{(\rho \ell)^2}{2^\lambda} + 3(\rho \ell)^2 \cdot (\text{MAX}(\epsilon_{\text{KE}}, \epsilon_{\text{KEM}}, \epsilon_{\text{NIKE}}) + \epsilon_{\text{SEXT}} + \epsilon_{\text{PRF}}).$$

**Proof.** It is straightforward to verify that two accepted oracles (of considered protocol) having matching sessions would generate the same session key. In the sequel, we wish to show that the adversary is unable to distinguish random value from the session key of any eCK-fresh oracle. Without loss of generality, we consider that the adversary chooses the test oracle $\pi_{s^*}^{\hat{A}}$ executed between owner $\hat{A}$ with its intended partner $\hat{B}$.

Next we introduce the notations which might be used in the proof. General speaking, we use the superscript $^*$ to highlight corresponding values processed in test oracle $\pi_{s^*}^{\hat{A}}$. Let party $\hat{D}$ denote the intended communication partner of oracle $\pi_{s^*}^{\hat{A}}$, where $\hat{D}$ could be any parties but $\hat{B}$, and let corresponding ciphertext received by oracle $\pi_{s^*}^{\hat{A}}$ be $C_{\hat{D}}$.

Applying the propositions 1 and 2 from [MO11b], the security proof can be given only under freshness cases $C_2$, $C_5$ and $C_6$ (see detail in Section 5.1.5). The proof proceeds in a sequence of games, following [Sho04, BR06]. The first game is the real security experiment, as assumed that there exists an adversary $\mathcal{A}$ that breaks the session key security of the proposed protocol. We then describe several intermediate games that step-wisely modify the original game. Finally we prove that (under the stated security assumptions), no adversary can break the security of the protocol.

\(^2\)The $\kappa$-min-entropy property of those keys is required by the strong randomness extractor SEXT, where $\kappa$ is a security parameter.
Let $S_\delta$ be the event that the adversary wins the security experiment under the Game $\delta$ and freshness cases in the set $\{C_2, C_5, C_6\}$. Let $\text{Adv}_\delta := \Pr[S_\delta] - 1/2$ denote the advantage of $A$ in Game $\delta$. Consider the following sequence of games.

**Game 0.** This is the original eCK game with adversary $A$ under freshness cases $C_2$, $C_5$ and $C_6$. Thus we have that

$$\Pr[S_0] = 1/2 + \epsilon = 1/2 + \text{Adv}_0.$$ 

**Game 1.** In this game, the challenger proceeds exactly like previous game, except that we add a abortion rule. The challenger raises event $\text{abort}_{\text{trans}}$ and aborts, if during the simulation either the ephemeral key $epk^*_i$ (outputted by KE) or $C^*_i$ (generated by KEM) replied by an oracle $\pi^*_i$ but it has been sample by another oracle $\pi^w_i$ or sent by adversary before. Since there are $\rho \ell$ such values would be sampled randomly by KEM or KE. Due to theirs security, the event $\text{abort}_{\text{trans}}$ occurs with probability $\Pr[\text{abort}_{\text{trans}}] \leq \left(\frac{\rho \ell}{2}\right)^2$ where $\lambda$ is a large integer as assumed before. We therefore have that

$$\text{Adv}_0 \leq \text{Adv}_1 + \frac{(\rho \ell)^2}{2^\lambda}.$$ 

Note that the transcript of protocol messages shared by two oracles having matching sessions is unique in this game, so that the adversary can’t replay any ciphertext or ephemeral key to result in two fresh oracles generating the same session key but without matching sessions.

**Game 2.** This game proceeds as previous game, but $C$ aborts if one of the following guesses fails: (i) the freshness case occurred to test oracle in the set $\{C_2, C_5, C_6\}$, (ii) the test oracle $\pi^*_A$, and (iii) its partner oracle $\pi^*_B$ in case $C_2$ or the intended communication party $\hat{B}$ in cases $C_5$ and $C_6$. Since there are 3 fresh related cases, $\ell$ parties at all and at most $\rho$ oracles for each party, then the probability that all guesses of $C$ are correct is at least $1/3(\rho \ell)^2$. Thus we have that

$$\text{Adv}_1 \leq 3(\rho \ell)^2 \cdot \text{Adv}_2.$$ 

In the following, we always assume that the challenger guesses correctly.

**Game 3.** This game is proceeded as previous game, but the challenger $C$ does the following modifications:

1. In the case $C_2$, replace the key $eK^*_A$ of test oracle $\pi^*_A$ and its partner oracle $\pi^*_B$ with random value $\tilde{eK}^*_A$. 

83
2. In the case $C5$, replace the NIKE key $ShK_{\hat{A},\hat{B}}$ with random value $\overline{ShK}_{\hat{A},\hat{B}}$ for all oracles having session participants $\hat{A}$ and $\hat{B}$. So that $\overline{ShK}_{\hat{A},\hat{B}}$ is used in place of either $\text{ShareKey}(\hat{A}, \hat{B})$ or $\text{ShareKey}(\hat{B}, \hat{A})$.

3. In the case $C6$, replace the KEM key $K^*_A$ encapsulated in the ciphertext $C^*_A$ generated by test oracle with random value $\overline{K^*_A}$. Whenever $C^*_A$ is received by an oracle $\pi_B$, the key $\overline{K^*_A}$ is used instead of $\text{KEM.DeCap}(sk_B, C^*)$.

If there exists an adversary $\mathcal{A}$ can distinguish the Game 3 from Game 2 then we can use it to construct an adversary $\mathcal{D}$ to break security of KE in the case $C2$ or NIKE in the case $C5$ or KEM in the case $C6$ respectively. Note that there will be at most one freshness case in the set $\{C2, C5, C6\}$ would occur in this game. Specifically, $\mathcal{D}$ simulates the challenger for $\mathcal{A}$ as previous game, in particular $\mathcal{D}$ would guess freshness case occurred to test oracle in the set $\{C2, C5, C6\}$, the test oracle and its partner oracle (if it exists). In the sequel, we assume that $\mathcal{D}$ guesses everything correctly as the challenger in the previous game, otherwise it aborts. Next $\mathcal{D}$ will modify the simulations to embed challenge values in terms of specific occurred cased as following:

1. **Case C2.** Given a challenge instance $(epk^*_1, epk^*_2, K^*_c)$ from KE security experiment (where $K^*_c$ is either a random value or the truth KE key in terms of challenge public keys $epk^*_1$ and $epk^*_2$), $\mathcal{D}$ does the following modifications:
   
   a) Set $epk^*_A := epk^*_1$ and $epk^*_B := epk^*_2$.
   
   b) Compute the key material for test oracle and its partner oracle as $eK^* := K^*_c$.

2. **Case C5.** $\mathcal{D}$ selects $(\hat{A}, \hat{B})$ as identities of NIKE-tested parties which have long-term public keys $p_k^*_1$ and $p_k^*_2$ (obtained via RegisterHonest query) and gets $K^*_c$ from Testniike$(\hat{A}, \hat{B})$ query in the NIKE security experiment, and it simulate the game for $\mathcal{A}$ with the following modifications:
   
   a) Set $pk_{\hat{A}} = p_k^*_1$ and $pk_{\hat{B}} = p_k^*_2$.
   
   b) Replace the shared key $ShK_{\hat{A},\hat{B}}$ with $K^*_c$ for oracles having both session participants $\hat{A}$ and $\hat{B}$.
   
   c) Generate the share key $ShK_{\hat{C},\hat{D}}$ using RevealKeyniike$(\hat{C}, \hat{D})$ from NIKE security experiment, when there is one and at most one party such that $\hat{C} \in \{\hat{A}, \hat{B}\}$ or $\hat{D} \in \{\hat{A}, \hat{B}\}$.

3. **Case C6.** Given a challenge instance $(C^*, pk^*, K^*_c)$ from KEM security experiment, $\mathcal{D}$ does the following modifications:
   
   a) Set $pk_{\hat{B}} = pk^*$ and the outgoing ciphertext of test oracle as $C^*_A := C^*$.
   
   b) Replace the KEM key $K^*_A$ with $K^*_c$ for test oracle and the oracles of $\hat{B}$ receiving the ciphertext $C^*_A$. 

84
c) Generate the key $K_D^\hat{D}$ of other oracles $\pi^t_B$ of party $\hat{B}$ using the decryption oracle $\mathcal{D}_{\mathcal{E}}(sk_{kem}^\hat{B}, \cdot)$, where $\hat{D}$ could be any parties but $\hat{B}$. Specifically, while receiving a ciphertext $C_D^\hat{D}$, oracle $\pi^t_B$ computes the $K_D^\hat{D} := \mathcal{D}_{\mathcal{E}}(sk_{kem}^\hat{B}, C_D^\hat{D})$.

As for the rest of the computations are done as the same as previous game.

To answer the RevealKey query and Test query for those modified oracles, $\mathcal{D}$ will use the changed key material to compute the final session key. With respect to the other queries, $\mathcal{D}$ simulates them honestly as the challenger in previous game using corresponding values chosen by herself. Without flipping the bit $b$, the Test-query is replied with the session key which is computed using modified key material. If $K^*_c$ is true key, then the simulation is equivalent to Game 2; otherwise the simulation is equivalent to Game 3. Finally, $\mathcal{D}$ returns what $A$ returns to KEM or KE or NIKE challenger in corresponding security experiment. If $A$ can distinguish the real key from the random value, that implies $\mathcal{D}$ break the KEM or KE or NIKE scheme under corresponding freshness case. Select the maximum advantage of adversary in each case, we therefore obtain that

$$\text{Adv}_2 \leq \text{Adv}_3 + \text{MAX}(\epsilon_{\text{KE}}, \epsilon_{\text{KEM}}, \epsilon_{\text{NIKE}}).$$

**Game 4.** We change this game from previous one by modifying the generation of output of SEXT in terms of freshness cases. Specifically, (i) in case C2 the value $eK^*$ of test oracle and its partner oracle(if it exists) is replaced with a uniform random value instead of the use of $\text{SEXT}(eK^*)$, (ii) in the case C5 the $ShK^A_{\hat{A}, B}^\prime$ is replaced with a uniform random value instead of the use of $\text{SEXT}(ShK^A_{\hat{A}, B}^\prime)$ for oracles having both session participants $\hat{A}$ and $\hat{B}$, (iii) and in the case C6, we replace $K^*_{\hat{A}}$ of test oracle with a random value and we do the same replacement for the oracles of $\hat{B}$ receiving the ciphertext $C^*_{\hat{A}}$ generated by test oracle. Those changes are possible since we have the fact either key $e\tilde{K}_c^*$ or key $ShK^A_{\hat{A}, B}$ or key $K^*_{\hat{A}}$ has been modified to be random value in previous game. By the security of the strong randomness extraction function, we have that

$$\text{Adv}_3 \leq \text{Adv}_4 + \epsilon_{\text{SEXT}}.$$

**Game 5.** In this game, we change function $\text{PRF}(ShK^A_{\hat{A}, B}^\prime, \cdot)$ in the case C5 or the function $\text{PRF}(K^*_{\hat{A}}^\prime, \cdot)$ in the case C6 or function $\text{PRF}(eK^*_{\hat{A}}^\prime, \cdot)$ in the case C2 to a truly random function for test oracle and its partner oracle (if it exists). We make use of the fact, that the at least one of those three secret seeds of the PRFs of test oracle is a truly random value due to the modification in previous game. If there exists a polynomial time adversary $A$ can distinguish the Game 5 from Game 4. Then we can construct an algorithm $B$ using $A$ to break the security of PRF. Exploiting the security of PRF, we
have that

$$\text{Adv}_4 \leq \text{Adv}_5 + \epsilon_{\text{PRF}}.$$ 

Note that in this game the session key returned by Test-query is totally a truly random value which is independent to the bit $b$ and any messages. Thus the advantage that the adversary wins this game is $\text{Adv}_5 = 0$.

Put together all probabilities from Game 0 to Game 5, we proved this theorem.

\[ \square \]

### 5.3 A One-round Two Party AKE Protocol under Standard Assumptions

In this section we present an eCK secure AKE protocol P1 in the standard model based on Bilinear Decisional Diffie-Hellman assumption. This protocol especially suits to the scenario that each party has registered only one public/private key pair. The proposed protocol relies on bilinear pairings, target collision resistant hash function family, and pseudo-random function family.

**Design Principle.** The construction of P1 can be seen as a concrete instantiation of GC-KKN scheme. We first observe that it is possible to merge those computations in KE, KEM and NIKE schemes if they work under the same algebraic groups. However the standard CKS-light secure NIKE scheme is rare so far. Our AKE construction is based on the pairing-based NIKE scheme [FHKP13] with slight modifications rather than the factoring-based NIKE scheme [FHKP13] because the latter requires a ‘uncertain’ interactive key registration protocol to ensure the validity of public key. Unlike the pairing-based NIKE scheme [FHKP13], we resort to target collision resistant hash function family instead of collision resistant chameleon hash function family to relax the assumption. This is possible because we could alternatively bind the identities of session participants with each session key using the pseudo-random function. In order to battle against CEK attacks, each party (including the attacker) is required to construct some kind of ‘tag’ based on specific weak Programmable Hash Function [HJK11] to encode consistency information on its chosen (either long-term or ephemeral) public keys. Those tags are particularly customized to be independent of any information about receivers, which enable our protocol to be able to run in the post-specified peer setting. However we have to make use of the pairing to provide a means of consistency checking that (both long-term and ephemeral) public keys coming from the adversary are in some sense of well-formed. Fortunately, those expensive consistency checks can be done on host machine.
5 One-round Two Party Authenticated Key Exchange

5.3.1 Protocol Description

The proposed protocol takes as input the following building blocks which are initialized respectively in terms of the security parameter κ:

- Symmetric bilinear groups \( \mathcal{PG} = (G, g, G_T, p, e) \) and along with random values \((u_1, u_2, u_3, u_4) \xleftarrow{\$} G\),

- a target collision resistant hash function \( \text{TCRHF}(hk_{\text{TCRHF}}, \cdot) : \mathcal{K}_{\text{TCRHF}} \times G \rightarrow \mathbb{Z}_p \), where \( hk_{\text{TCRHF}} \xleftarrow{\$} \text{TCRHF}.\text{KGen}(\kappa) \), and

- a pseudo-random function family \( \text{PRF}(\cdot, \cdot) : \mathbb{Z}_T \times \mathbb{D}_{\text{PRF}} \rightarrow \mathcal{K}_{\text{AKE}} \).

The variable \( \text{pms} \) stores public system parameters \( \text{pms} := (\mathcal{PG}, \{u_i\}_{1 \leq i \leq 4}, hk_{\text{TCRHF}}) \).

\[
\begin{align*}
\hat{A} & \quad \text{pk}_{\hat{A}} = (A, t_A) := (g^n, (u_4^h, u_3^h, u_2^n)) \\
& \quad \hat{B} \quad \text{pk}_B = (B, t_B) := (g^b, (u_4^h, u_3^h, u_2^n)) \\
& \quad x \xleftarrow{\$} \mathbb{Z}_p^*, X := g^x \\
& \quad h_X := \text{TCRHF}(X) \\
& \quad t_X := (u_4^h, u_3^h, u_2^n)^x \\
& \quad h_Y := \text{TCRHF}(Y) \\
& \quad t_Y := (u_4^h, u_3^h, u_2^n)^y \\
& \quad \hat{A} || A || t_A|| X || t_X || B || t_B || Y || t_Y \\
& \quad \hat{B} || B || t_B|| Y || t_Y \\
& \quad \text{reject if some values recorded in sid} \\
& \quad \beta_{\hat{A}} := e(u_1, BY) \\
& \quad k := \beta_{\hat{A}}^{-1} \\
& \quad \text{accept } k_c := \text{PRF}(k, \text{sid}) \\
& \quad \text{sid} := \\
& \quad \text{ACCEPT if some values recorded in sid} \\
& \quad \beta_B := e(u_1, AX) \\
& \quad k := \beta_B^{-1} \\
& \quad \text{accept } k_c := \text{PRF}(k, \text{sid})
\end{align*}
\]

**Figure 5.3:** Pairing-based AKE Protocol under Standard Assumptions

### Long-term Key Generation and Registration:

On input \( \text{pms} := (\mathcal{PG}, \{u_i\}_{1 \leq i \leq 4}, hk_{\text{TCRHF}}) \), a party \( \hat{A} \) may run an efficient algorithm \( (sk_{\hat{A}}, \text{pk}_{\hat{A}}, \emptyset) \xleftarrow{\$} \text{OR2AKE.KGen(\text{pms}, \hat{A})} \) to generate the long-term key pair as: \( sk_{\hat{A}} = a \xleftarrow{\$} \mathbb{Z}_p^*, \text{pk}_{\hat{A}} = (A, t_A) \) where \( A = g^a \), \( t_A := (u_4^h, u_3^h, u_2^n)^a \) and \( h_A = \text{TCRHF}(A) \). Please note that we allow arbitrary key registration, i.e. the adversary is able to query \( \text{EstablishParty}(\hat{A}, \text{pk}_{\hat{A}}, \emptyset) \).
with $\text{pf}_A = \emptyset$.

**Protocol Execution**: On input $\text{pms} := (PG, \{u_i\}_{1 \leq i \leq 4}, h_{\text{TCRHF}})$, the protocol between parties $A$ and $B$ proceeds as following which is also depicted in the Figure 5.3.

1. Upon activation a new session, the party $\hat{A}$ performs the steps: (a) choose an ephemeral private key $x \overset{\$}{\leftarrow} \mathbb{Z}_p^*$; (b) compute ephemeral public key $X := g^x$; (c) compute $h_X := \text{TCRHF}(X)$, and $t_X := (u_4^{h_X^2} u_3^{h_X} u_2)^x$; (d) send $(\hat{A}, X, t_X)$ to $\hat{B}$.

2. Upon activation a new session, the party $\hat{B}$ performs the steps: (a) choose an ephemeral private key $y \overset{\$}{\leftarrow} \mathbb{Z}_p^*$; (b) compute the ephemeral public key $Y := g^y$; (c) compute $h_Y := \text{TCRHF}(Y)$, and $t_Y := (u_4^{h_Y^2} u_3^{h_Y} u_2)^y$; (d) send $(\hat{B}, Y, t_Y)$ to $\hat{A}$.

3. Upon receiving $(\hat{B}, Y, t_Y)$, the party $\hat{A}$ does the following: (a) compute $h_Y := \text{TCRHF}(Y)$ and $h_B := \text{TCRHF}(B)$, and reject if either $e(t_Y, g) \neq e(u_4^{h_Y^2} u_3^{h_Y} u_2, Y)$ or $e(t_B, g) \neq e(u_4^{h_B^2} u_3^{h_B} u_2, B)$; (b) set session identifier $\text{sid} := \hat{A}||A||X||t_X||Y||t_Y||\hat{B}||B||Y||t_Y$, and reject the session if some values recorded in $\text{sid}$ are identical. (c) compute $\beta_A := e(u_1, BY)$; (d) compute $k := \beta_A^{a+x}$ and session key as $k_e := \text{PRF}(k, \text{sid})$.

4. Upon receiving $(\hat{A}, X, t_X)$, the party $\hat{B}$ does the following: (a) compute $h_X := \text{TCRHF}(X)$ and $h_A := \text{TCRHF}(A)$, and reject if either $e(t_X, g) \neq e(u_4^{h_X^2} u_3^{h_X} u_2, X)$ or $e(t_A, g) \neq e(u_4^{h_A^2} u_3^{h_A} u_2, A)$; (b) set session identifier $\text{sid} := \hat{A}||A||X||t_X||Y||t_Y||\hat{B}||B||Y||t_Y$, and reject the session if some values recorded in $\text{sid}$ are identical. (c) compute $\beta_B := e(u_1, AX)$; (d) compute $k := \beta_B^{b+y}$, session key as $k_e := \text{PRF}(k, \text{sid})$.

**Implementation and Session States**: We assume that only the ephemeral private keys $x$ (resp. $y$) would be stored in the state variable $st$, and these states are the maximum states which can be compromised by adversary. This can be guaranteed by performing the computations of key material $k$ and session key $k_e$ (i.e. (steps 3.(d) and step 4.(d)) on secure device.

**Remark 1**. We assume that the messages in $\text{sid}$ are ordered by round, and within each round lexicographically by the identities of the purported senders. Please note that the computation cost at secure device of the above scheme is dominated by only one exponentiation and one PRF. For instance, the implementer could only compute the key material $k := \beta_A^{a+x}$ and finally session key on secure device, where $\beta_A := e(u_1, BY)$ is computed on host machine. We stress that all other expensive operations (such as pairing) can be done on host machine which is normally more powerful than secure device. Of course one could execute the whole protocol on secure device, but then it will be much less efficient. Instead we only assume that the most necessary computations are done on secure device without pairing operations. The secure device could take as
input the ephemeral key and prepared key materials (e.g. the \(x\) and \(e(u_1, BY)\)), and it outputs the generated session key using key derivation function PRF. Moreover, those parameters used to compute those tags are particularly customized to be independent of any information about session participants. This enables the protocol to be able to perfectly run in the post-specified peer setting.

### 5.3.2 Performance Improvement

In this section, we consider the issue on improving the efficiency of proposed protocol P1. It is obvious that the consistency checks on both long-term and ephemeral keys consume much computational cost which requires four pairing operations. Thus how to reduce the cost on those consistency checks is our major concern.

We first introduce an alternative consistency checking algorithm which is derived from the similar technique in [KG06, KG09] used to improve the efficiency of identity-based KEM scheme. The idea is to merge consistency checks on incoming Diffie-Hellman keys. In the new consistency check algorithm, a party \(\hat{A}\) on receiving \((\hat{B}, B, t_B, Y, t_Y)\) may perform the following steps:

1. Choose two random values \(\theta_1, \theta_2 \leftarrow \mathbb{Z}_p^*\).
2. Reject the session if \(e(t_B^{\theta_1} t_Y^{\theta_2} g, g) \neq e((u_4^{h_B} u_3^{h_B} u_2^{\theta_1}), B) e( (u_4^{h_B} u_3^{h_B} u_2^{\theta_1})^{\theta_2}, Y)\).

We claim that the combined consistency check equation implies that all received tags are consistent. In order to prove our argument we define functions \(\Delta_1(t_Y) := \frac{e(u_2^{h_Y} u_3^{h_B} u_4^{h_B} u_2^{\theta_1})^{\theta_1}}{e(t_Y g)}\) and \(\Delta_2(t_B) := \frac{e(u_2^{h_B} u_3^{h_B} u_4^{h_B} B)}{e(t_B g)}\). Obviously, we have \(\Delta_1(t_Y) = \Delta_2(t_B) = 1\) if and only if \(t_Y, t_B\) are consistent. Consequently, for random values \(\theta_1, \theta_2 \leftarrow \mathbb{Z}_p^*\), function \(\Delta_1(t_Y)^{\theta_1} (\Delta_2(t_B))^{\theta_2}\) evaluates to 1 if \(t_Y, t_B\) are consistent and to a random group value in \(G_T\) otherwise. This alternative consistency check algorithm substitutes one multiple-exponentiation for one pairing operation. Note that the above technique could be extended to merge more than two consistency check equations that would dramatically improve the efficiency of consistency check procedure, e.g. the consistency check operations in our upcoming one-round tripartite protocols in Section 6.2 and one-round group AKE protocols in Section 6.3. However, the random values \(\theta_1\) and \(\theta_2\) would be included into the returns of \textit{StateReveal} query since the consistency check algorithm is executed on host machine.

Furthermore, please note that a party \(\hat{A}\) has to do consistency check on long-term key in every sessions that might be wasteful. An alternative solution could make the Certificate Authority to check the consistency of long-term public key during key registration procedure. In this way, it might reduce two pairing operations for protocol execution and also the number of public key. To register a public key \(pk_{\hat{A}} = A\), each party \(\hat{A}\) should at least prove the consistency via tag \(t_A\). Then the public key \(A\) is registered if \(e(t_A, g) = e(A, u_4^{h_A} u_3^{h_B} u_2^{\theta_2})\). Thus this check would be done only once at CA. The
downside of this approach is that it might increase the burden of CA. In particular, the tag $t_A$ is required while querying the $\text{EstablishParty}(\hat{A}, pk_{\hat{A}}, pf_{\hat{A}})$ in the security game, i.e. $pf_{\hat{A}} = t_A$.

### 5.3.3 Security Analysis

We show the security of proposed protocol in our strong security model.

**Theorem 5.** Assume each ephemeral key chosen during key exchange has bit-size $\lambda \in \mathbb{N}$. Suppose that the BDDH problem is $(t, \epsilon_{\text{BDDH}})$-hard in the symmetric bilinear groups $\mathcal{P}_G$, the TCRHF is $(t, \epsilon_{\text{TCRHF}})$-secure target collision resistant hash function family, and the PRF is $(q_{\text{prf}}, t, \epsilon_{\text{PRF}})$-secure pseudo-random function family. Then the proposed protocol is $(t', \epsilon)$-eCK-secure in the sense of Definition 27 with $t' \approx t$, $q_{\text{prf}} \geq 2$ and

$$\epsilon \leq \frac{(\rho\ell)^2}{2^\lambda} + \epsilon_{\text{TCRHF}} + 3(\rho\ell)^2 \cdot (\epsilon_{\text{BDDH}} + \epsilon_{\text{PRF}}).$$

**Proof.** First of all, it is easy to verify that the proposed protocol satisfying the correctness in the sense of Definition 23. Namely two accepted oracles with matching sessions would generate the same session key. In the sequel, we are going to show that the adversary is unable to distinguish random value from the session key of any eCK-fresh oracle. Without loss of generality, we consider that the adversary chooses the test oracle $\pi_{s \hat{A}}$ executed between owner $\hat{A}$ with its intended partner $\hat{B}$.

Next we introduce the notations which might be used in the proof. Let party $\hat{C}$ denote the intended partner of oracle $\pi_{s \hat{A}}$ where $\hat{C}$ could be any parties but $\hat{A}$ in the security game and has long-term public key $C = g^c$, and let $W = g^w$ denote the ephemeral public key received by oracle $\pi_{s \hat{A}}$. In a similar way, oracle $\pi_{t \hat{B}}$ has intended party denoted by $\hat{D}$ with long-term public key $D = g^d$, and the received ephemeral key by $\pi_{t \hat{B}}$ is $N = g^n$. Meantime the ephemeral keys generated by oracles $\pi_{s \hat{A}}$ and $\pi_{t \hat{B}}$ are $X = g^x$ and $Y = g^y$ respectively. we use the superscript "*" to highlight corresponding values processed by the test oracle and its partner oracle (if it exists), say the ephemeral key $X^*$ generated by oracle $\pi_{s \hat{A}}^*$.

Applying the propositions 1 and 2 from [MO11a], the security proof can be given only under freshness cases $C2$, $C5$ and $C6$ (see Section 5.1.5 of this Chapter). The proof proceeds in a sequence of games, following [Sho04, BR06]. The first game is the real security experiment, as assumed that there exists an adversary $\mathcal{A}$ that breaks the session key security of the proposed protocol. We then describe several intermediate games that step-wisely modify the original game. Finally we prove that (under the stated security assumptions), no adversary can break the security of the protocol. Let $S_\delta$ be the event that the adversary wins the security experiment under the Game $\delta$ and freshness cases in the set $\{C2, C5, C6\}$. Let $\text{Adv}_\delta := Pr[S_\delta] - 1/2$ denote the advantage of $\mathcal{A}$ in Game $\delta$. 
Game 0. This is the original game running with adversary $A$ under freshness cases $C_2, C_5$ and $C_6$. The system parameters are chosen honestly by challenger as protocol specification. Meanwhile, the challenger chooses six uniform random values $(r_1, r_2, r_3, r_4) \leftarrow \mathbb{Z}_p^*$, and sets $u_1 := g^{r_1}, u_2 := g^{r_2}, u_3 := g^{r_3}$ and $u_4 := g^{r_4}$ as public parameters. All queries are simulated honestly in terms of protocol specification. Thus we have that

$$\Pr[S_0] = 1/2 + \epsilon.$$ 

Game 1. In this game, the challenger proceeds exactly like previous game, except that we add an abort rule. The challenger raises event $abort_{eph}$ and aborts, if during the simulation an ephemeral key replied by an oracle $\pi_i$ but it has been sampled by another oracle or sent by adversary before. Since there are $\rho \ell$ such ephemeral keys would be sampled uniform randomly from $\{0, 1\}^\lambda$. Thus, the event $abort_{eph}$ occurs with probability at least $\Pr[abort_{eph}] \leq \frac{\rho^2 \ell^2}{2^\lambda}$. We have that

$$\text{Adv}_0 \leq \text{Adv}_1 + \frac{\rho^2 \ell^2}{2^\lambda}.$$ 

Note that the ephemeral key chosen by each oracle is unique in this game.

Game 2. In this game we want to make sure that the received ephemeral keys are correctly formed. Technically, we add an abort condition, namely the challenger proceeds exactly as before, but raises an event $abort_{hash}$ and aborts if there exist two distinct (either ephemeral or long-term) public keys $W$ and $N$ such that $\text{TCRH}(W) = \text{TCRH}(N)$. Obviously the $\Pr[abort_{hash}] \leq \epsilon_{\text{TCRH}}$, according to the security property of underlying hash function. Thus we have

$$\text{Adv}_1 \leq \text{Adv}_2 + \epsilon_{\text{TCRH}}.$$ 

Game 3. This game proceeds as previous game, but $C$ aborts if one of the following guesses fails: (i) the freshness case occurred to test oracle in the set $\{C_2, C_5, C_6\}$, (ii) the test oracle $\pi_A^{s_i}$, and (iii) its partner oracle $\pi_B^{t_i}$ in case $C_2$ or the intended communication partner $\hat{B}$ in cases $C_5$ and $C_6$. Since there are 3 fresh related cases, $\ell$ parties at all and at most $\rho$ oracles for each party, then the probability that all guesses of $C$ are correct is at least $1/3(\rho \ell)^2$. Thus we have that

$$\text{Adv}_2 \leq 3(\rho \ell)^2 \cdot \text{Adv}_3.$$
Game 4. In this game, we want to reduce the security of proposed protocol to the hardness of BDDH problem. Please first note that there are at least two uncompromised (either long-term and ephemeral) Diffie-Hellman (DH) keys which are used by test oracle to generate its key material $k^*$, in terms of certain freshness case. As otherwise the test oracle is no longer eCK-fresh. We call such guessed two uncompromised DH keys as target DH keys.

This game is proceeded as previous game, but the challenger $C$ replaces the key material $k^*_s$ with random value $\tilde{k}^*_s$ for oracles $\{\pi^s_i : i \in [\ell], s \in [\rho]\}$ which satisfy the following conditions:

- The $k^*_s$ is computed involving the two target DH keys which are guessed by $C$ for test oracle, and
- These two target DH keys used by $\pi^s_i$ are from two distinct parties.

Of course if two oracles satisfying above two conditions which have matching sessions, then we could use the same modified key material to generate corresponding session key. The second condition is necessary, because the adversary can easily result in one oracle receiving DH keys from certain party which are all uncompromised DH keys via e.g. Send query EstablishParty queries. On the other side, the first condition cannot exclude the event that the DH keys from certain party are all chosen (or revealed) by adversary. In this case, the adversary can compute the session key of such oracle and we cannot change the key material of that oracle any more. The above two conditions ensure that the changed key materials of oracles can not be trivially generated by adversary.

If there exists an adversary $A$ can distinguish between Game 4 and Game 3 then we can use it to construct a distinguisher $D$ to solve the BDDH problem as follows. Given a BDDH challenge instance $(g, g^\nu, g^\omega, g^\xi, \Gamma) \in \mathbb{G}^3 \times \mathbb{G}^T$, $D$'s goal is to determine whether $\Gamma$ equals to $e(g, g)^{\nu \omega \xi}$ or a random element, where $g$ is a group generator of $\mathbb{G}$. More specifically, $D$ simulates the challenger for $A$ as previous game but with the following modifications based on its correct guesses (otherwise it aborts). Meanwhile, let $p(h) = p_0 + p_1 h + p_2 h^2$ be a polynomial of degree 2 over $\mathbb{Z}_p^*$. The detail form of this polynomial will be discussed in the simulation based on specific freshness case. Let $q(h) = q_0 + q_1 h + q_2 h^2$ be a random polynomial of degree 2 over $\mathbb{Z}_p^*$.

1. Case $C2$. In this case $D$ does the following modifications:
   a) Set $X^* := g^\nu$, $Y^* := g^\omega$ and $u_1 := g^\xi$.
   b) Compute the key material of test oracle and its partner oracle as:
      - $k^*_A = k^*_B := \Gamma \cdot e(u_1, BY^*)^a \cdot e(u_1, X^*)^b$.
   c) Compute those tags of test oracle and its partner oracle as:
      - $t^*_X := (X^*)r_2(h_X^*)^2 + r_3 h_X^* + r_4$, where $h_X^* = \text{TCRF}(X^*)$. 

5 One-round Two Party Authenticated Key Exchange

- \( t^*_Y := (Y^*)^{r_2(h^*_Y)^2+r_3 h^*_Y + r_4} \), where \( h^*_Y = \text{TCRHF}(Y^*) \).

2. **Case C5.** In this case, \( \mathcal{D} \) does the following modifications:
   
a) Set \( u_1 := g^r, A := g^w \), and \( B := g^h \).
   
b) Set polynomial \( p(h) \) to satisfy that \( p(h) = (h - h_A)(h - h_B) \), where \( h_A = \text{TCRHF}(A) \), \( h_B = \text{TCRHF}(B) \).
   
c) Set \( u_{i+2} = u_1^{p_i}g^B \) for \( 0 \leq i \leq 2 \).
   
d) Compute the \( t_A := A^{q(h_A)} \) and \( t_B := B^{q(h_B)} \).
   
e) Replace the value \( e(u_1, A)^b \) with \( \Gamma \) when computing the key material \( k \) of oracles \( \pi_A^x \) and \( \pi_B^x \) which involve both long-term public keys \( A \) and \( B \) (including the test oracle \( \hat{\pi}_A^x \)), more specifically:
     - \( k_A^x := \Gamma \cdot e(u_1, BW)^x \cdot e((\frac{t_w}{W \cdot h_W}), A) \), where \( h_W = \text{TCRHF}(W) \).
     - \( k_B^x := \Gamma \cdot e(u_1, AN)^y \cdot e((\frac{t_N}{N \cdot h_N}), B) \), where \( h_N = \text{TCRHF}(N) \).
   
f) Compute the secret key material for other oracles \( \pi_A^x \) and \( \pi_B^x \) of party \( \hat{A} \) and \( \hat{B} \):
     - \( k_A^x := e(u_1, CW)^x \cdot e((\frac{t_w}{W \cdot h_W}), A) \cdot e((\frac{t_c}{C \cdot h_C}), A) \), where \( h_W = \text{TCRHF}(W) \) and \( h_C = \text{TCRHF}(C) \).
     - \( k_B^x := e(u_1, ND)^y \cdot e((\frac{t_N}{N \cdot h_N}), B) \cdot e((\frac{t_d}{D \cdot h_D}), B) \), where \( h_W = \text{TCRHF}(N) \) and \( h_D = \text{TCRHF}(D) \).

3. **Case C6.** In this case, \( \mathcal{D} \) does the following modifications:
   
a) Set \( X^* := g^r, B := g^w \) and \( u_1 := g^h \).
   
b) Set polynomial \( p(h) \) to satisfy that \( p(h) = (h - h_X^*)(h - h_B) \), where \( h_X^* = \text{TCRHF}(X^*) \), \( h_B = \text{TCRHF}(B) \).
   
c) Set \( u_{i+2} = u_1^{p_i}g^B \) for \( 0 \leq i \leq 2 \).
   
d) Compute \( t_B := B^{q(h_B)} \) and \( t^*_X := B^{q(h_X^*)} \).
   
e) Compute the key material \( k^*_A \) of test oracle \( \hat{\pi}_A^x \) and \( k^*_B \) of oracles \( \hat{\pi}_B^x \) having session states \( t^*_X \) and \( X^* \), and corresponding tag \( t^*_X \) as:
     - \( k_A^x := \Gamma \cdot e((\frac{t_w}{W \cdot h_W}), X^*) \cdot e(u_1, BW^x)^x \), where \( h_W = \text{TCRHF}(W^x) \).
     - \( k_B^x := \Gamma \cdot e(u_1, DX^*)^y \cdot e((\frac{t_p}{P \cdot h_D}), B) \), where \( h_D = \text{TCRHF}(D) \).
   
f) Change the computation of secret key material \( k_B^x \) of other oracles of \( \hat{B} \) as:
     - \( k_B^x := e(u_1, DN)^y \cdot e((\frac{t_p}{P \cdot h_D}), (\frac{t_N}{N \cdot h_N}))(\frac{t_n}{N \cdot h_N}), B) \).

Those modified tags are consistent with the original form. We make use of the fact there is no collision on those hash values due to the result of previous game. To answer
the **RevealKey** query for those modified oracles, the $\mathcal{D}$ will use the changed key material (e.g. $k_B^\epsilon$) to compute the final session key as protocol specification. With respect to the other queries, the $\mathcal{D}$ simulates them honestly as the challenger using corresponding values chosen by herself. Without flipping the bit $b$, the **Test**-query is replied with the session key which is computed using modified key material. Based on the condition that all guesses by $\mathcal{D}$ are correct, if $\Gamma = e(g, g)^\nu \omega \xi$, then the simulation is equivalent to Game 3; otherwise the simulation is equivalent to Game 4. At the end of the simulation, $\mathcal{D}$ returns what $\mathcal{A}$ returns to BDDH challenger. If $\mathcal{A}$ can distinguish the real key from the random value, that implies $\mathcal{D}$ solves the BDDH problem. We therefore obtain that

$$\text{Adv}_3 \leq \text{Adv}_4 + \epsilon_{\text{BDDH}}.$$ 

**Game 5.** In this game, we change function $\text{PRF}(\tilde{k}_A^\epsilon, \cdot)$ to a truly random function for test oracle and its partner oracle (if it exists). We make use of the fact, that the secret seed $\tilde{k}_A^\epsilon$ of test oracle is a truly random value. If there exists a polynomial time adversary $\mathcal{A}$ can distinguish the Game 5 from Game 4. Then we can construct an algorithm $\mathcal{B}$ using $\mathcal{A}$ to break the security of $\text{PRF}$. Exploiting the security of $\text{PRF}$, we have that

$$\text{Adv}_4 \leq \text{Adv}_5 + \epsilon_{\text{PRF}}.$$ 

Note that in this game the session key returned by **Test**-query is totally a truly random value which is independent to the bit $b$ and any messages. Thus the advantage that $\mathcal{A}$ wins this game is $\text{Adv}_5 = 0$.

Sum up the probabilities from Game 0 to Game 5, we proved this theorem.

□

**5.4 Comparisons**

We summarize the comparisons of some existing well-known eCK secure protocols without random oracles in the Table 5.1 from the following perspectives: (i) the knowledge of peer’s public key (peer setting), which would affect e.g. the round efficiency as aforementioned; (ii) the special design trick used to guarantee corresponding security; (iii) the security assumptions; (iv) number of long-term (LL) and ephemeral (Eph) keys which may affect the storage requirement, computation cost and communication cost; (v) overall computation cost of considered protocol; (vi) the computation cost on secure device (SD) and (vii) the communication round between host machine and secure device (HS). In the table, ‘Exp’ denotes the exponentiation and ‘ME’ denotes multi-exponentiations, ‘Pair’ denotes pairing evaluation, ‘(T)CR’ denotes (target) collision-resistant hash, ‘DDH’ denotes the decisional Diffie-Hellman assumption, ‘FAC’ denote
factoring assumption, ‘EXT’ denotes the strong randomness extractor, and ‘TPRF’ denotes the twisted-PRF trick. Recall that P1 denotes the protocol in Section 5.3.

<table>
<thead>
<tr>
<th>Peer setting</th>
<th>Design trick</th>
<th>Security assumptions</th>
<th>LL (pk,sk)</th>
<th>Eph (pk,sk)</th>
<th>Overall cost</th>
<th>SD cost</th>
<th>HS round</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Oka07]</td>
<td>post TPRF</td>
<td>DDH, CR PRF</td>
<td>(2,4)</td>
<td>(3,2)</td>
<td>3.Exp, 1.ME</td>
<td>2.Exp, 1.ME</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.CR, 1.PRF</td>
<td>1.ME, 1.PRF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.PRF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[MO11a]</td>
<td>post -</td>
<td>DDH, CR PRF</td>
<td>(4,5)</td>
<td>(3,2)</td>
<td>3.Exp, 1.ME</td>
<td>1.ME, 1.PRF</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.CR, 1.PRF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[FSXY12]</td>
<td>pre TPRF</td>
<td>DDH, FAC EXT, TCR PRF</td>
<td>(3,1)</td>
<td>(3,2)</td>
<td>6.Exp, 2.ME</td>
<td>3.Exp, 2.ME</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.TCR, 3.EXT</td>
<td>2.TCR, 4.PRF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5 PRF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GC-KKN § 5.2</td>
<td>pre -</td>
<td>DDH, FAC EXT, TCR PRF</td>
<td>(6,2)</td>
<td>(3,2)</td>
<td>7.Exp, 2.ME</td>
<td>2.Exp, 1.ME</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.TCR, 3.PRF</td>
<td>2.TCR, 2.PRF</td>
<td></td>
</tr>
<tr>
<td>P1 § 5.3</td>
<td>post -</td>
<td>BDDH, TCR PRF</td>
<td>(2,1)</td>
<td>(2,1)</td>
<td>2.Exp, 4.ME</td>
<td>1.Exp, 1.PRF</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.Pair, 2.TCR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.PRF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Comparisons among one-round 2AKE protocols in the standard model

We instantiate the FSXY protocol with the factoring-based KEM [HK09] (also suggested by the authors of FSXY) and with the DDH-based ElGamal KEM. For comparison, we also instantiate our GC-KKN for instance with the same factoring-based KEM as FSXY, with the factoring-based NIKE scheme [FHKP13] and with the traditional Diffie-Hellman key exchange (DHKE) [DH76] under DDH assumption.

In addition we assume without loss of generality the FSXY, Okamoto [Oka07] and MO [MO11a] schemes are realized with secure device to protect critical session states. For simplicity and security, we just assume the twisted-PRF trick used by both Okamoto and FSXY schemes is implemented on secure device to avoid any leakage on its output. Similarly all computations related to twisted-PRF trick are assumed to be executed on secure device too, such as the whole IND-CCA secure KEM (both encapsulation/decapsulation algorithms) used by FSXY. Otherwise the FSXY protocol is not secure in the eCK model.

We further remark that both FXSY and Okamoto protocols need at least two rounds to exchange information between host machine and secure device. In the first round the host machine may send randomness to secure device and get back the ephemeral key (or ciphertext of KEM.EnCap algorithm); and in the second round, the host machine may send intermediate values to secure device and obtain the final session key from secure device. Therefore, our GC-KKN scheme is more HS-round efficient than those schemes, since only one such round is required. Although optimized P1 requires four pairing operations, it enjoys the most succinct structure and the most efficient algorithm on secure device.
6 One-round Group Authenticated Key Exchange

The situation where three or more parties share a secret key is often called group (conference) keying. A group authenticated key exchange protocol (GAKE) allows a set of parties communicating over public network to create a common shared key that is ensured to be known only to those entities. GAKE protocols are essentially generalized from two party authenticated key exchange (2AKE) protocols to the case of multiple parties. However, this brings new challenges not only in the design but also in the analysis of GAKE protocols. In this chapter we focus on the construction for one-round GAKE protocols which are expected to be secure in the g-eCK model.

Paper. This Chapter is based on the paper working with Yong Li: Strongly Secure One-round Group Authenticated Key Exchange in the Standard Model, which is currently in submission. Yong and I first provide general results to simplify the security analysis for one-round group AKE protocols in the g-eCK model in Section 6.1.5. On the second I proposed a concrete construction for g-eCK secure one-round 3AKE protocol without random oracles in Section 6.2. Yong can I came up with the idea of building g-eCK secure one-round group AKE protocols. Meanwhile I was responsible for the detail construction and security analysis in Section 6.3.

6.1 Background

6.1.1 Notions for One-round GAKE

We first present a generic definition of one-round group authenticated key exchange (ORGAKE) to allow us to describe our generic result for this class of protocols. The notions of one-round GAKE can be generalized from one-round two party AKE in Section 5.1.1. Let \( GD := ((ID_1, pk_{ID_1}), \ldots, (ID_n, pk_{ID_n})) \) be a list which is used to store the public information of a group of parties formed as tuple \((ID_i, pk_{ID_i}), \) where \( n \) is the size of the group members which intend to share a key and \( pk_{ID_i} \) is the public key of party \( ID_i \in IDS (i \in [n]) \). Let \( T \) denote the transcript storing the messages sent and received by a protocol instance at a party which are sorted orderly. A general PKI-based ORGAKE protocol may consist of four polynomial time
algorithms (ORGAKE.Setup, ORGAKE.KGen, ORGAKE.MF, ORGAKE.SKG) with following semantics:

- \( pms \leftarrow \text{Setup}(\kappa) \): This algorithm takes as input a security parameter \( \kappa \) and outputs a set of system parameters storing in a variable \( pms \).

- \((sk_{ID}, pk_{ID}, pf_{ID}) \leftarrow \text{ORGAKE.KGen}(pms, ID)\): This algorithm takes as input system parameters \( pms \) and a party’s identity \( ID \), and outputs a pair of long-term private/public key \((sk_{ID}, pk_{ID}) \in \{PK, SK\}\) for party \( ID \) and a non-interactive proof for \( pk_{ID} \) (which is required during key registration).

- \( m_{ID_1} \leftarrow \text{ORGAKE.MF}(pms, sk_{ID_1}, r_{ID_1}, GD)\): This algorithm takes as input system parameters \( pms \) and the sender \( ID_1 \)’s secret key \( sk_{ID_1} \), a randomness \( r_{ID_1} \leftarrow \mathcal{R}_{\text{ORGAKE}} \) and the group information variable \( GD \), and outputs a message \( m_{ID_1} \in \mathcal{M}_{\text{ORGAKE}} \) to be sent, where \( \mathcal{R}_{\text{ORGAKE}} \) is the randomness space and \( \mathcal{M}_{\text{ORGAKE}} \) is the message space.\(^1\)

- \( K \leftarrow \text{ORGAKE.SKG}(pms, sk_{ID_1}, r_{ID_1}, GD, T)\): This algorithm takes as the input system parameters \( pms \) and \( ID_1 \)’s secret key \( sk_{ID_1} \), a randomness \( r_{ID_1} \leftarrow \mathcal{R}_{\text{ORGAKE}} \) and the group information \( GD \) and a transcript \( T \) orderly recorded all protocol messages exchanged\(^2\), and outputs session key \( K \in \mathcal{K}_{\text{ORGAKE}} \), where \( \mathcal{K}_{\text{ORGAKE}} \) is the session key space.

For correctness, we explicitly require that, on input the same group description \( GD = ((ID_1, pk_1), \ldots, (ID_n, pk_n)) \) and transcript \( T \), the algorithm ORGAKE.SKG satisfies the constraint: \( \text{ORGAKE.SKG}(pms, sk_{ID_1}, r_{ID_1}, GD, T) = \text{ORGAKE.SKG}(pms, sk_{ID_i}, r_{ID_i}, GD, T) \) where \( sk_{ID_i} \) is the secret key of a party \( ID_i \in GD \) who generates randomness \( r_{ID_i} \in \mathcal{R}_{\text{ORGAKE}} \) for \( i \in [n] \).

Besides these algorithms, each protocol might consist of other steps such as long-term key registration and message exchange, which should be described by each protocol independently.

### 6.1.2 Motivations

The most recent FMSU protocol [FMSU12] satisfies g-eCK security (that is known to be one of the strongest security model for GAKE). However its security proof is given in the random oracle model (ROM) under a specific strong assumption, i.e. gap Bilinear Diffie-Hellman (GBDH) assumption [BSNS05]. So far we are not aware of previous GAKE protocols being able to achieve g-eCK security in the standard model. Hence,

---

\(^1\)We remark that the parameter \( GD \) of algorithm ORGAKE.MF is only optional, which can be any empty string if specific protocol compute the message without knowing any information about its intended partners.

\(^2\)The detail order needs to be specified by each protocol.
one of the open problems in research on GAKE is to construct a secure scheme in the g-eCK model under standard assumptions without resorting to random oracles. In this work, we are therefore motivated to seek efficient solutions to construct g-eCK secure GAKE protocol in the standard model. Another important motivation of this paper is to try to simplify the security proof for GAKE protocols under the g-eCK model from the perspective of reducing the freshness cases that need to prove. Since under the g-eCK model, the freshness cases (see detail in Section 5.1.5) are related to the group size which are not a small amount.

6.1.3 Contributions

We solve the above open problems by starting from 3AKE. In Section 6.2, we firstly give a concrete construction for one-round 3AKE protocol that is g-eCK secure in the standard model under standard assumptions. The proposed protocol is based on bilinear groups, target collision resistant hash function family, and pseudo-random function family.

In order to withstand active attackers (e.g. chosen ephemeral key or long-term public key attacks), each (either long-term or ephemeral) public key is required to be associated with some kind of ‘tag’ which is used to verify the consistency of corresponding public key. Those tags are particularly customized using specific weak Programmable Hash Functions [HJK11] for ephemeral key and long-term key respectively, whose output lies in a pairing group. Interestingly the proposed protocol is built to be able to run without knowing any priori information about its partners’ long-term public key. We make use of the pairing to provide a means of consistency checking that (both long-term and ephemeral) public keys coming from the adversary are in some sense of well-formed. Due to those tags, all distinct public keys are mutually independent. Hence an active adversary is not able to lead non-partnered fresh sessions to generate co-related session keys. In particular, any active adversaries have to leave session key related secret information in those tags which can be extracted and exploited by the challenger (during the proof simulation) using corresponding trapdoor secret, e.g. the exponents of the group elements used to compute the tags. In order to facilitate the security analysis of 3AKE protocols in the g-eCK model, we introduce propositions to formally reduce fourteen freshness cases (which cover all freshness cases for one-round 3AKE) to four freshness cases. Then it is only necessary to prove the security of considered protocol under the reduced four freshness cases. Any g-eCK security analyzers for one-round 3AKE might benefit from these results. We then provide a succinct and rigorous game-based security proof by reducing the g-eCK security of proposed 3AKE protocol in the standard model to break the cubic Bilinear Decisional Diffie-Hellman (CBDDH) assumption which is slightly modified from the Bilinear Decisional Diffie-Hellman (BDDH) assumption [Jou04].
In the latter we present a GAKE scheme with constant maximum group size in Section 6.3 following the construction idea of 3AKE. Nevertheless the proposed GAKE scheme is based on the symmetric multilinear map which is first postulated by Boneh and Silverberg [BS02]. Informally speaking, the symmetric multilinear groups are equipped with a $n$-multilinear maps $me : \mathbb{G}^n \rightarrow \mathbb{G}_T$ where $n \geq 2$ is an integer, $\mathbb{G}$ is a multiplicative cyclic group of large prime order $p$ and $\mathbb{G}_T$ is the target group with the same order. Most recently, Garg, Gentry and Halvai [GGH13, GGH12] introduced a surprising candidate mechanism that would approximate multilinear maps in discrete-logarithm hard groups. Their result may open the opportunity to implement constructions using a multilinear map abstraction in practice. We prove g-eCK security of our scheme in the standard model under a natural multilinear generalization of the CBDDH assumption which is called $n$-Multilinear Decisional Diffie-Hellman Assumption ($nMDDH$). In particular we give a general game-based security proof for our proposed GAKE scheme which is given under any polynomial number of freshness cases. This general proof is applicable when the group size of our GAKE protocol ranges from $2$ to $n + 1$, that also implies the concrete security proof of our 3AKE protocol in Section 6.2.2 and 2AKE protocol in Section 5.3.3.

6.1.4 Related Works

One import research direction in GAKE field is to construct secure one-round protocol due to its appealing bandwidth-efficient. A prominent example is the pairing based tripartite protocol introduced by Antoine Joux [Jou00, Jou04] which extends the classical two-party Diffie-Hellman KE protocol [DH76] to the three party case. Joux’s protocol is unauthenticated and subject to well known man-in-the-middle attacks. Since then tripartite authenticated key exchange (3AKE) as a special form of group key exchange has drawn much attention of researchers. Several attempts, e.g. [ARP03, LL05, LLPL07, MSU09, FMSU12], have been made to improve the original protocol. As pointed out by Manulis et al. (MSU) [MSU09] the solutions [ARP03, LL05, LLPL07] fail to achieve implicit authentication, and the authors introduced a new 3AKE protocol. The proposed MSU protocol consists of two communication rounds in which the first round is used to establish the session key and the key confirmation steps are done in the second round. MSU protocol was proved secure in a very strong model which is extended from the two party eCK07 model [LLM07] to group case. In this paper we call this model as g-eCK+ model which satisfies almost all known security requirements for GAKE. Of course the MSU protocol [MSU09] can be executed within one round, i.e. without the key confirmation steps. However then it is only able to provide weak perfect forward secrecy (wPFS), due to Krawczyk’s generic PFS attack [Kra05] to one round key exchange protocols. The wPFS was first studied by Krawczyk in literature [Kra05] and was latter formalized for one round GAKE protocols in the
g-eCK model [FMSU12]. We remark that the only difference between g-eCK+ model and g-eCK model is that the latter formalizes the wPFS instead of PFS. Thus in this thesis we mainly focus on the g-eCK model for one-round GAKE. To provide g-eCK security, Fujioka et al. (FMSU) [FMSU12] generalized previous 3AKE protocols into one framework based on admissible polynomials developed from [FS11] which yields many further one round 3AKE protocols.

In 2009, Gorantla et al. [GBGM09] extend the technique of Boyd et al. [BCNP08] to the group setting and present a generic one-round GKE protocol based on a primitive called multi key encapsulation mechanism (mKEM) [Sma04]. However GBGM protocol is unable to provide g-eCK security. Consider the following attack scenario that the test oracle generates the ciphertext $C^*$ of an IND-CCA secure mKEM scheme and all other ciphertexts received by test oracle are generated by adversary. Therefore any adversary who obtains the ephemeral secret states used in generating the encapsulation key $C^*$ can compute the session key and trivially win the g-eCK security game.

6.1.5 Simplify the Security Proof for One-round GAKE in the g-eCK model

We first show how to reduce the complexity of the security proof of any one-round 3AKE protocol with the above form in the g-eCK model. To prove the security of a protocol in the g-eCK model, it is necessary to show the proof under all possible freshness cases formulated by Definition 28. Let oracle $\pi_s^*$ be the test oracle with intended partner $\hat{B}$ and $\hat{C}$ for instance. If any adversary breaks the indistinguishability security property of an OR3AKE protocol, then at least one of the following fresh events and related cases must occur:

1. **Event 0**: There are oracles $\pi^s_B$ and $\pi^r_C$, such that $\pi^s_A$ has matching session to $\pi^r_B$ and to $\pi^r_C$ respectively. Then we have the following cases:

   - Case 1 (C1): The adversary did not query $\text{StateReveal}(\pi^s_A)$, nor $\text{StateReveal}(\pi^r_B)$ nor $\text{StateReveal}(\pi^r_C)$.
   - Case 2 (C2): The adversary did not query $\text{Corrupt}(\hat{A})$, nor $\text{StateReveal}(\pi^r_B)$ nor $\text{StateReveal}(\pi^r_C)$.
   - Case 3 (C3): The adversary did not query $\text{StateReveal}(\pi^s_A)$, nor $\text{StateReveal}(\pi^r_B)$ nor $\text{Corrupt}(\hat{C})$.
   - Case 4 (C4): The adversary did not query $\text{Corrupt}(\hat{A})$, nor $\text{StateReveal}(\pi^r_B)$ nor $\text{Corrupt}(\hat{C})$.
   - Case 5 (C5): The adversary did not query $\text{StateReveal}(\pi^s_A)$, nor $\text{Corrupt}(B)$ nor $\text{StateReveal}(\pi^r_C)$. 

• Case 6 (C6): The adversary did not query \( \text{Corrupt}(\hat{A}) \), nor \( \text{Corrupt}(\hat{B}) \) nor \( \text{StateReveal}(\pi^*_A) \).

• Case 7 (C7): The adversary did not query \( \text{Corrupt}(\hat{A}) \), nor \( \text{Corrupt}(\hat{B}) \) nor \( \text{Corrupt}(\hat{C}) \).

• Case 8 (C8): The adversary did not query \( \text{StateReveal}(\pi^*_A) \), nor \( \text{Corrupt}(\hat{B}) \) nor \( \text{Corrupt}(\hat{C}) \).

2. Event 1: There is an oracle \( \pi^*_F \) such that \( \pi^*_A \) and \( \pi^*_F \) have matching sessions but there is no oracle at \( \hat{D} \) having matching session to \( \pi^*_A \), where \( \hat{F}, \hat{D} \in \{ \hat{B}, \hat{C} \} \) and \( \hat{D} \neq \hat{F} \). Then we have the cases:

• Case 9 (C9): The adversary did not query \( \text{StateReveal}(\pi^*_A) \), nor \( \text{StateReveal}(\pi^*_F) \) nor \( \text{Corrupt}(\hat{D}) \).

• Case 10 (C10): The adversary did not query \( \text{Corrupt}(\hat{A}) \), nor \( \text{StateReveal}(\pi^*_F) \) nor \( \text{Corrupt}(\hat{D}) \).

• Case 11 (C11): The adversary did not query \( \text{Corrupt}(\hat{A}) \), nor \( \text{Corrupt}(\hat{F}) \) nor \( \text{Corrupt}(\hat{D}) \).

• Case 12 (C12): The adversary did not query \( \text{StateReveal}(\pi^*_A) \), nor \( \text{Corrupt}(\hat{F}) \) nor \( \text{Corrupt}(\hat{D}) \).

3. Event 2: \( \pi^*_A \) has no matching session, we have the following cases:

• Case 13 (C13): The adversary did not query \( \text{Corrupt}(\hat{A}) \), nor \( \text{Corrupt}(\hat{B}) \) nor \( \text{Corrupt}(\hat{C}) \).

• Case 14 (C14): The adversary did not query \( \text{StateReveal}(\pi^*_A) \), nor \( \text{Corrupt}(\hat{B}) \) nor \( \text{Corrupt}(\hat{C}) \).

Let ‘nRS’ denote the situation that the adversary didn’t issue \( \text{StateReveal} \) query to specific oracle, and ‘nC’ denote the situation adversary did not ask \( \text{Corrupt} \) query to corresponding party (e.g. the owner of certain oracle). In the Table 6.1, we summarize the above freshness cases might be occurred in each event.

<table>
<thead>
<tr>
<th>Event 0</th>
<th>Event 1</th>
<th>Event 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1 (C1)</td>
<td>( \pi^*_A ) nRS</td>
<td>( \pi^*_F ) nRS</td>
</tr>
<tr>
<td>Case 2 (C2)</td>
<td>nC nRS</td>
<td>nRS nC</td>
</tr>
<tr>
<td>Case 3 (C3)</td>
<td>nRS nRS</td>
<td>nC nC</td>
</tr>
<tr>
<td>Case 4 (C4)</td>
<td>nC nRS</td>
<td>nC nC</td>
</tr>
<tr>
<td>Case 5 (C5)</td>
<td>nRS nC</td>
<td>nRS nC</td>
</tr>
<tr>
<td>Case 6 (C6)</td>
<td>nC nC</td>
<td>nC nC</td>
</tr>
<tr>
<td>Case 7 (C7)</td>
<td>nC nC</td>
<td>nC nC</td>
</tr>
<tr>
<td>Case 8 (C8)</td>
<td>nRS nC</td>
<td>nC nC</td>
</tr>
</tbody>
</table>

| Case 9 (C9) | nRS nRS | nC nC |
| Case 10 (C10) | nC nRS | nC nC |
| Case 11 (C11) | nC nC | nC nC |
| Case 12 (C12) | nRS nC | nC nC |

| Case 13 (C13) | nC nC | nC nC |
| Case 14 (C14) | nRS nC | nC nC |

Table 6.1: Freshness cases in each event
In order to complete the proof, we must provide the security proofs under all fourteen cases that might be tiresome. However we introduce the following general propositions to facilitate the proof of any OR3AKE protocols in the form of the above description. Our goal is to reduce the freshness cases which have the similar restrictions on adversary’s queries.

**Proposition 3.** If adversary $A_1 (t_1, \epsilon_{A_1})$-breaks the g-eCK security of a OR3AKE protocol $\Sigma$ in case $C_2$, then there exists adversary $A_2$ who can $(t_2, \epsilon_{A_2})$-breaks the g-eCK security of $\Sigma$ in case $C_3$ (C5), such that $t_1 \approx t_2$ and $\epsilon_{A_1} = \epsilon_{A_2}$.

**Proof.** Intuitively, in cases $C_2$ and $C_3$ (C5), the test oracle has matching sessions, then the adversary could selects either an oracle $\pi^s_A$ or its partners $\pi^r_B$ or $\pi^r_C$ as test oracle since $\pi^s_A$, $\pi^r_B$, and $\pi^r_C$ will compute the same session key. In both cases the adversary reveals the states of two oracles and corrupt a party. We show the security reduction as follows. $A_2$ interacts with the AKE challenger $C$ and tries to break the security of considered protocol under freshness case $C_3$ (C5). It runs $A_1$ as subroutine and responds all oracle queries except for the test oracle. When $A_1$ issues the Test query to $\pi^s_A$, $A_2$ selects the matching partner $\pi^r_B$ as the test oracle. When $A_2$ receives the real session key or random key, $A_2$ sends it to $A_1$. If $A_1$ outputs a bit, $A_2$ outputs the same bit. Note that $A_2$ can issue StateReveal($\pi^s_A$) since it has matching session to the ‘test oracle’ (i.e. $\pi^r_B$) from the view of $A_2$. $A_2$ can issue Corrupt($\hat{A}$) since from the view of $A_2$ the party $\hat{A}$ is the intended partner of its ‘test oracle’. So $A_2$ can correctly respond to all queries issued by $A_1$. Therefore, if $A_1$ breaks the security of the considered protocol in case $C_2$, $A_2$ wins the game in case $C_3$ (C5) with the same advantage as $A_1$’s. □

**Proposition 4.** If adversary $A_1 (t_1, \epsilon_{A_1})$-breaks the g-eCK security of a OR3AKE protocol $\Sigma$ in case $C_3$ (C5), then there exists adversary $A_2$ who can $(t_2, \epsilon_{A_2})$-breaks the g-eCK security of $\Sigma$ in case $C_9$, such that $t_1 \approx t_2$ and $\epsilon_{A_1} = \epsilon_{A_2}$.

**Proof.** This proof is similar to the proof of Proposition 3. $A_2$ interacts with an AKE challenger $C$ and tries to break the security of considered protocol under freshness case $C_9$. It runs $A_1$ as subroutine and responds all the oracle queries except for the test oracle. $A_1$ issues the Test query to oracle $\pi^s_A$ which has a matching session to oracle $\pi^r_D$ and to $\pi^r_B$ in the view of $A_1$. Note that the oracle $\pi^r_D$ is simulated by challenger $C$, but the oracle $\pi^r_B$ is simulated by $A_2$ on behalf of $\hat{D}$ for $A_1$ (since $A_2$ is the challenger of $A_1$). The message generated by oracle $\pi^r_B$ is either generated by $A_2$ or obtained from an oracle simulated by $C$, which depends on $A_2$’s choice. So that when $A_2$ receives the real session key or random key, $A_2$ sends it to $A_1$. If $A_1$ outputs a bit, $A_2$ outputs the same bit. $A_2$ can respond to any oracle queries issued by $A_1$ since the restricted oracle queries are equivalent. Therefore, $A_2$ breaks the security of the considered protocol in

102
case C9 if \( \mathcal{A}_1 \) wins the game in case C3 (C5).

**Proposition 5.** If adversary \( \mathcal{A}_1 (t_1, \epsilon_{\mathcal{A}_1}) \)-breaks the g-eCK security of a OR3AKE protocol \( \Sigma \) in case C7, then there exists adversary \( \mathcal{A}_2 \) who can \( (t_2, \epsilon_{\mathcal{A}_2}) \)-breaks the g-eCK security of \( \Sigma \) in case C11. If such adversary \( \mathcal{A}_2 \) exists, then there exists adversary \( \mathcal{A}_3 \) who can \( (t_3, \epsilon_{\mathcal{A}_3}) \)-breaks the g-eCK security of \( \Sigma \) in case C13. We have that \( t_1 \approx t_2 \approx t_3 \) and \( \epsilon_{\mathcal{A}_1} = \epsilon_{\mathcal{A}_2} = \epsilon_{\mathcal{A}_3} \).

**Proof.** This proof is similar to the proofs of Proposition 3 and Proposition 4. \( \mathcal{A}_2 \) runs \( \mathcal{A}_1 \) as subroutine and responds all the oracle queries except for the test oracle. \( \mathcal{A}_1 \) issues the Test query to oracle \( \pi^*_A \) which has a matching session to oracle \( \pi^*_F \) and to \( \pi^*_D \) (the view of \( \mathcal{A}_1 \)). \( \mathcal{A}_2 \) could select \( \pi^*_F \) as test oracle which is possible since it has matching session to \( \pi^*_A \). So that when \( \mathcal{A}_2 \) receives the real session key or random key, \( \mathcal{A}_2 \) sends it to \( \mathcal{A}_1 \). If \( \mathcal{A}_1 \) outputs a bit, \( \mathcal{A}_2 \) outputs the same bit. \( \mathcal{A}_2 \) can respond to any oracle queries issued by \( \mathcal{A}_1 \) since the restricted oracle queries for those adversaries are equivalent. Therefore, \( \mathcal{A}_2 \) breaks the security of the considered protocol in case C11 if \( \mathcal{A}_1 \) wins the game in case C7. Analogously we have the reduction from C11 to C13, since the restricted oracle queries are the same to adversaries. \( \square \)

**Proposition 6.** If adversary \( \mathcal{A}_1 (t_1, \epsilon_{\mathcal{A}_1}) \)-breaks the g-eCK security of a OR3AKE protocol \( \Sigma \) in case C4 (C6), then there exists adversary \( \mathcal{A}_2 \) who can \( (t_2, \epsilon_{\mathcal{A}_2}) \)-breaks the g-eCK security of \( \Sigma \) in case C8. If such adversary \( \mathcal{A}_2 \) exists, then there exists adversary \( \mathcal{A}_3 \) who can \( (t_3, \epsilon_{\mathcal{A}_3}) \)-breaks the g-eCK security of \( \Sigma \) in case C12. If such adversary \( \mathcal{A}_3 \) exists, then there exists adversary \( \mathcal{A}_4 \) who can \( (t_4, \epsilon_{\mathcal{A}_4}) \)-breaks the g-eCK security of \( \Sigma \) in case C14. We have that \( t_1 \approx t_2 \approx t_3 \approx t_4 \) and \( \epsilon_{\mathcal{A}_1} = \epsilon_{\mathcal{A}_2} = \epsilon_{\mathcal{A}_3} = \epsilon_{\mathcal{A}_4} \).

**Proof.** This proof is similar to the proofs of Proposition 3, Proposition 4 and Proposition 5. Thus we omit the detail here for avoid repetition. \( \square \)

The above reductions routes are informally depicted in the Figure 6.1.

\[
C2 \rightarrow C3(C5) \rightarrow C9 \\
C7 \rightarrow C11 \rightarrow C13 \\
C4(C6) \rightarrow C8 \rightarrow C12 \rightarrow C14
\]

**Figure 6.1:** Reductions of g-eCK-Freshness for One-round Tripartite AKE

Due to the above reductions, one could prove the security of any one-round 3AKE protocol in the g-eCK model only under freshness cases C1, C9, C13 and C14. This would be dramatically simplify the security proof. In the sequel, we call these freshness cases need to write proof as target freshness cease.
Towards Lower Bound of Target Freshness Cases for the Proof of One-round GAKE with Arbitrary Group Size in the g-eCK Model. In order to make the proof for one-round GAKE protocol in the g-eCK model to be more tight, we might also need to do the analogous reductions about the freshness cases as it is done for OR3AKE. However, we might not be able to formally do so in a short page when the group size \( n \in \mathbb{N} \) is a large integer, in which the total number of the freshness cases would be very large. So that we can only make certain conjecture for the lower bound of target freshness cases for the proof of GAKE protocol with arbitrary group size \( n \) in the eCK Model.

**Conjecture 1.** For any one-round group AKE protocol with members \( n + 1 \), we have \( n + 2 \) freshness cases that require proof simulation.

**Proof.** Please first note that each freshness case consists of two main aspects: (i) the status (e.g. the number) of matching sessions, (ii) the restrictions of adversary’s queries. Since there are at most \( n + 1 \) parties in a protocol instance for which can be queried either **Corrupt** or **StateReveal**. It is not hard to see that there are \( n + 1 \) distinct events for the status of matching sessions, i.e. matching sessions of test oracle vary between \( n \) and \( 0 \). Let ‘Event i’ \((0 \leq i \leq n)\) denote the situation that the test oracle has \( n - i \) matching sessions (we use the similar representation approach as three party case). In Event i, there is \( 2^{n-i+1} \) distinct freshness cases, because either the test oracle or each oracle having matching session to test oracle has two distinct cases, i.e. it can be either corrupted or revealed states by adversary. Collect the number of freshness cases in each event, we have total number of freshness cases \( 2(2^{n+1} - 1) = \sum_{i=0}^{n} 2^{n-i+1} \).

From the reductions for three party case, we know that freshness cases having analogous or the same query restrictions could be somehow reduced. So that we only need to do proof simulations for these ‘target’ freshness cases having distinct query restrictions. We observe that there are \( n + 2 \) such target freshness cases. Since each event would contribute one distinct freshness case, except for the Event n in which event there is two distinct freshness cases related to test oracle. Namely in Event i \((i \neq n)\) that the test oracle has \( n - i \) matching sessions, we have the distinct freshness case as: The adversary did not query **StateReveal** to these \( n - i \) matching sessions (of test oracle) and did not query **Corrupt** to these \( i \) parties which have no matching session to test oracle. \( \square \)

### 6.2 A Tripartite AKE Protocol from Bilinear Maps

In this section we present a three party one-round AKE protocol based on symmetric bilinear groups, a target collision resistant hash function and a pseudo-random function family. The requirements for underlying building blocks are standard, the proposed protocol provides g-eCK security without random oracles.
**Design Principle.** The challenge here is that we have to simultaneously cope with chosen identity and long-term public key (CIDPK) attacks (modeled by EstablishParty query) and chosen ephemeral key (CEK) attacks (modeled by Send query) in presence of strong adversaries who can reveal non-trivial session states (via StateReveal query) and even compromise the long-term keys of participants (via Corrupt query). As any other attacks in the g-eCK model, we say an adversary successfully launches a CEK attack or CIDPK attack on considered AKE protocol $\Sigma$ in the g-eCK model, if it $(t, \epsilon)$-breaks the g-eCK security of $\Sigma$. Informally speaking the CIDPK attack addresses the situation that the adversary registers dishonest identity and long-term public key and tries to subvert the security, e.g. obtain information about honest user’s secret key via small sub-group attacks [LL97]. The CEK attack addresses the situation that the adversary tries to manipulate the session key via exchanged ephemeral keys of her own choice. To deal with these complicated situations, we have to set up a proof simulation for our construction in the g-eCK model that is able to simulate all queries ‘appropriately’.

Our main idea is to make use of the weak (3,poly)-PHF [HJK11] to resist with not only CEK attacks but also CIDPK attacks under the g-eCK model. This is possible, since there are at most three (either long-term or ephemeral) public keys will not be compromised by adversary. However, we can’t efficiently construct the protocol based on BDDH assumption. Because in a BDDH challenge instance, all DH keys are distinct to each other. Consider the most awkward case that there are at most three uncorrupted parties, each of which may possess a long-term key generated by a BDDH challenge value. Thus we might need at least three different weak (3,poly)-PHFs to plug in all BDDH challenge values in order to simulate the session keys correctly for all those uncompromised oracles. To avoid this inefficient setting, the CBDDH assumption might be a perfect alternative choice. We could simultaneously embed CBDDH challenge value into those uncompromised DH keys and the parameters of weak (3,poly)-PHF in the security proof.

### 6.2.1 Protocol Description

We describe the protocol in terms of the following three parts: Setup, long-term key generation and key registration, protocol execution, one could think of the general algorithms defined in Section 5.1.5 are implied in specific part.

**Setup:** The proposed protocol takes as input the following building blocks which are initialized respectively in terms of the security parameter $\kappa \in \mathbb{N}$:

- Symmetric bilinear groups $\mathcal{PG} = (\mathcal{G}, g, \mathcal{G}_T, p, e) \xleftarrow{\$} \text{PG.Gen}(1^\kappa)$ and a set of random values $\{u_i\}_{0 \leq i \leq 3} \xleftarrow{\$} \mathcal{G}$,

- a target collision resistant hash function $\text{TCRHF}(hk_{\text{TCRHF}}, \cdot) : \mathcal{K}_{\text{TCRHF}} \times \mathcal{G} \rightarrow \mathbb{Z}_p$, where $hk_{\text{TCRHF}} \xleftarrow{\$} \text{TCRHF.Gen}(1^\kappa)$, and
Long-term Key Generation and Registration: On input $pms := (\mathcal{PG}, \{u_i\}_{0 \leq i \leq 3}, \mathit{hk}_{\mathit{TCRHF}})$, a party $\hat{A}$ may run an efficient key generation algorithm $(sk_{\hat{A}}, pk_{\hat{A}}, \emptyset) \xleftarrow{\$} \mathcal{ORGAKE.KGen}(pms, \hat{A})$ to generate the long-term key pair as: $sk_{\hat{A}} = a \xleftarrow{\$} \mathbb{Z}_p^*$, $pk_{\hat{A}} = (A, t_A)$ where $A = g^a$, $t_A = (u_0u_1^A u_2^A u_3^A)^a$ and $h_A = \mathit{TCRHF}(A)$. Please note that we allow arbitrary key registration, i.e. the adversary is able to query $\mathcal{EstablishParty}(\hat{A}, pk_{\hat{A}}, \emptyset)$ with $\mathit{pf}_{\hat{A}} = \emptyset$.  

Figure 6.2: One-round Tripartite AKE Protocol
Protocol Execution: On input the system parameter $pms$, the protocol among parties $\hat{A}$, $\hat{B}$ and $\hat{C}$ is executed as following, which is also informally depicted in the Figure 6.2.

1. Upon activating a new session with participants $(\hat{A}, \hat{B}, \hat{C})$, the party $\hat{A}$ first chooses an ephemeral private key $x \leftarrow \mathbb{Z}_p^*$ and compute ephemeral public key $X := g^x$. Next $\hat{A}$ computes $h_X := \text{TCRHF}(X)$, and $t_X := (u_0 u_1^h h_2^k h_3^y x)^z$. To the end $\hat{A}$ broadcasts messages $(\hat{A}, X, t_X)$ to $\hat{B}$ and $\hat{C}$.

2. Upon activating a new session with participants $(\hat{A}, \hat{B}, \hat{C})$, the party $\hat{B}$ first chooses an ephemeral private key $y \leftarrow \mathbb{Z}_p^*$ and compute ephemeral public key $Y := g^y$. Next $\hat{B}$ computes $h_Y := \text{TCRHF}(Y)$, and $t_Y := (u_0 u_1^v h_2^k h_3^z y)^z$. To the end $\hat{B}$ broadcasts messages $(\hat{B}, Y, t_Y)$ to $\hat{A}$ and $\hat{C}$.

3. Upon activating a new session with participants $(\hat{A}, \hat{B}, \hat{C})$, the party $\hat{C}$ first chooses an ephemeral private key $z \leftarrow \mathbb{Z}_p^*$ and compute ephemeral public key $Z := g^z$. Next $\hat{C}$ computes $h_Z := \text{TCRHF}(Z)$, and $t_Z := (u_0 u_1^z h_2^z h_3^3 z)^z$. To the end $\hat{C}$ broadcasts messages $(\hat{C}, Z, t_Z)$ to $\hat{A}$ and $\hat{B}$.

4. Upon receiving $(\hat{B}, Y, t_Y)$ and $(\hat{C}, Z, t_Z)$, the party $\hat{A}$ sets identifier $\text{id} := \hat{A}|\text{id}|t_B||x||t_Y||y||t_Z||Z||t_Z$ and rejects the session if some values recorded in $\text{id}$ are identical. Next $\hat{A}$ computes $h_B = \text{TCRHF}(B)$, $h_C = \text{TCRHF}(C)$, $h_Y = \text{TCRHF}(Y)$ and $h_Z = \text{TCRHF}(Z)$ and rejects the session if either $e(t_B, g) \neq e(u_0 u_1^h h_2^b h_3^b, B)$ or $e(t_C, g) \neq e(u_0 u_1^c h_2^c h_3^c, B)$ or $e(t_Y, g) \neq e(u_0 u_1^y h_2^y h_3^y, Y)$ or $e(t_Z, g) \neq e(u_0 u_1^z h_2^z h_3^z, Z)$. Finally, $\hat{A}$ computes $k := e(BY, CZ)^{a+x}$ and session key $k_e := \text{PRF}(k, \text{id})$.

5. Upon receiving $(\hat{A}, X, t_X)$ and $(\hat{C}, Z, t_Z)$, the party $\hat{B}$ sets identifier $\text{id} := \hat{B}|\text{id}|t_B||x||t_X||y||t_Z||Z||t_Z$ and rejects the session if some values recorded in $\text{id}$ are identical. Next $\hat{B}$ computes $h_A = \text{TCRHF}(A)$, $h_C = \text{TCRHF}(C)$, $h_X = \text{TCRHF}(X)$ and $h_Z = \text{TCRHF}(Z)$ and rejects the session if either $e(t_A, g) \neq e(u_0 u_1^h h_2^b h_3^b, A)$ or $e(t_C, g) \neq e(u_0 u_1^c h_2^c h_3^c, A)$ or $e(t_X, g) \neq e(u_0 u_1^x h_2^x h_3^x, X)$ or $e(t_Z, g) \neq e(u_0 u_1^z h_2^z h_3^z, Z)$. Finally, $\hat{B}$ computes $k := e(AX, CZ)^{b+z}$ and session key $k_e := \text{PRF}(k, \text{id})$.

6. Upon receiving $(\hat{A}, X, t_X)$ and $(\hat{B}, Y, t_Y)$, the party $\hat{C}$ sets identifier $\text{id} := \hat{A}|\text{id}|t_B||x||t_X||y||t_Y||C||t_Z||Z||t_Z$ and rejects the session if some values recorded in $\text{id}$ are identical. Next $\hat{C}$ computes $h_B = \text{TCRHF}(B)$, $h_C = \text{TCRHF}(C)$, $h_X = \text{TCRHF}(X)$ and $h_Y = \text{TCRHF}(Y)$ and rejects the session if either $e(t_B, g) \neq e(u_0 u_1^h h_2^b h_3^b, B)$ or $e(t_C, g) \neq e(u_0 u_1^c h_2^c h_3^c, C)$ or $e(t_X, g) \neq e(u_0 u_1^x h_2^x h_3^x, X)$ or $e(t_Y, g) \neq e(u_0 u_1^y h_2^y h_3^y, Y)$. Finally, $\hat{A}$ computes $k := e(A, BY)^{a+x}$ and session key $k_e := \text{PRF}(k, \text{id})$.

Implementation and Session States: We assume that the maximum states of party $\hat{A}$ allowing for leakage consist of ephemeral private key $x$ (resp. $y$ and $z$ for
6 One-round Group Authenticated Key Exchange

parties \( \hat{B} \) and \( \hat{C} \) – namely those values would be stored in the state variable \( st \) of each oracle at any time. For example this can be guaranteed by performing the computations for \( k \) and \( k_e \) on secure device. Note that the all pairing operations including \( e(BY, CZ) \) can be done on host machine.

In a nutshell, other non-trivial states, e.g. the secret exponent \( c + z \) and key material \( k \), should be carefully protected. We stress that it is not allowed to simultaneously leak the ephemeral private key say \( z \) and secret key material say \( e(AX, BY)^{c+z} \) to any attackers. Otherwise the protocol is insecure in the g-eCK model. Since such attacker can simply replay the ephemeral key say \( X \) generated by test session owned by \( \hat{A} \) to any session of \( \hat{C} \) and extract non-trivial secret \( e(AX, BY)^c \) from the knowledge of \( z \) and \( k \), where \( Y \) could be chosen by the attacker on behalf of \( \hat{B} \). Then it can break the security by sending any ephemeral keys \( Z' = g^{z'} \) and \( Y \) of her own choice to test session which generates the session key \( PRF(e(AX, BY)^{c+z'}, sid) \). Analogously the leakage of ephemeral private key \( z \) and corresponding secret exponent \( c + z \) would lead to the expose of private key \( c \). As well the leakage of only exponent \( c + z \) would enable adversary to launch infinite replay attacks.

We remark that our scheme can satisfy perfect forward secrecy by increase key confirmation procedures in an extra round, but the protocol then would become less efficient. We leave this problem for future work, that is to construct secure one-round GAKE protocols in the g-eCK+ model [MSU09]. Moreover, we could make use of the similar technique as introduced in Section 5.3.2 to merge those consistency checks as following:

1. Upon receiving \( (\hat{C}, \hat{A}, A, t_A, X, t_X) \) and \( (\hat{C}, C, t_C, Z, t_Z) \), the party \( \hat{A} \) computes
   \[
   U_B := u_0 u_1^{h_B} u_2^{h_B} u_3^{h_B}, \quad U_C := u_0 u_1^{h_C} u_2^{h_C} u_3^{h_C}, \quad U_Y := u_0 u_1^{h_Y} u_2^{h_Y} u_3^{h_Y} \quad \text{and} \quad U_Z := u_0 u_1^{h_Z} u_2^{h_Z} u_3^{h_Z}.
   \]

2. \( \hat{A} \) chooses four random values \( \theta_1, \theta_2, \theta_3, \theta_4 \) \( \xleftarrow{\$} \mathbb{Z}_p^* \).

3. \( \hat{A} \) rejects the session if \( e(\theta_1 B, \theta_2 C, \theta_3 Y, \theta_4 Z, g) \neq e(U_B^{\theta_1}, B) e(U_C^{\theta_2}, C) e(U_Y^{\theta_3}, Y) e(U_Z^{\theta_4}, Z) \).

Analogously we claim the above combined consistency check equation implies that all received tags are consistent. The proof of this argument can be extended from the proof in Section 5.3.2, and we therefore omit it here. Of course, the CA could do the consistency checks on long-term public key for each party in key registration protocol. Then one could set the tag, e.g. \( t_A \) corresponding to \( A \), as proof \( pf = t_A \) while asking \( \text{EstablishParty}(\hat{A}, A, t_A) \) query.

6.2.2 Security Analysis

We show the security of proposed protocol in the g-eCK model.
Theorem 6. Assume each ephemeral key chosen during key exchange has bit-size $\lambda \in \mathbb{N}$. Suppose that the CBDDH problem is $(t, \epsilon_{\text{CBDDH}})$-hard in the symmetric bilinear groups $\mathcal{P}G$, the TCRHF is $(t, \epsilon_{\text{TCRHF}})$-secure target collision resistant hash function family, and the PRF is $(q_{\text{prf}}, t, \epsilon_{\text{PRF}})$-secure pseudo-random function family. Then the proposed protocol is $(t', \epsilon)$-g-eCK-secure in the sense of Definition 29 with $t' \approx t$, $q_{\text{prf}} \geq 3$ and
\[
\epsilon \leq \frac{(\rho \ell)^2}{2^\lambda} + \epsilon_{\text{TCRHF}} + 4(\rho \ell)^3 \cdot (\epsilon_{\text{CBDDH}} + \epsilon_{\text{PRF}}).
\]

Proof. It is straightforward to see that two oracles accept with matching sessions would compute the same session key. Namely the proposed protocol is correct. In the sequel, we wish to show that the adversary is unable to distinguish random value from the session key of any fresh oracle. Without loss of generality, we consider that the adversary chooses the test oracle $\pi_{s \hat{A}}$ executed with its intended partners $\hat{B}$ and $\hat{C}$.

Next we introduce the notations which might be used in the proof. Let the ephemeral keys generated by oracles $\pi_{s \hat{A}}$, $\pi_{t \hat{B}}$ and $\pi_{t \hat{C}}$ are $X = g^x$, $Y = g^y$ and $Z = g^z$ respectively. We use the superscript $\ast$ to highlight corresponding values processed by the test oracle and its partner oracles (if they exist), say the ephemeral key $X\ast$ generated by oracle $\pi_{s \ast \hat{A}}$. Let $D = g^d$, $F = g^f$, $W = g^w$ and $V = g^v$ denote the Diffie-Hellman (DH) keys received by an oracle $\pi_{s i}$ ($i \in [\ell]$) which are used to compute the session key, where these DH keys could be either ephemeral or long-term key.

To complete the proof of Theorem 6, we only need to prove the advantage of the adversary is negligible under target freshness cases $C_1$, $C_9$, $C_{13}$ and $C_{14}$, due to the reductions (by Proposition 3, Proposition 4, Proposition 5 and Proposition 6.) in Section 5.1.5. The proof proceeds in a sequence of games, following [Sho04, BR06]. Let $S_\delta$ be the event that the adversary wins the security experiment in Game $\delta$ and freshness cases in the set $\{C_1, C_9, C_{13}, C_{14}\}$. Let $\text{Adv}_\delta := \Pr[S_\delta] - 1/2$ denote the advantage of $A$ in Game $\delta$.

Game 0. This is the original game with adversary $A$. The system parameters are chosen honestly by challenger as protocol specification. Meanwhile, the challenger chooses four uniform random values $\{r_i\} \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ for $0 \leq i \leq 3$, and sets $u_i := g^{r_i}$ as public parameters. Thus we have that
\[
\Pr[S_0] = 1/2 + \epsilon = 1/2 + \text{Adv}_0.
\]

Game 1. In this game, the challenger proceeds exactly like previous game, except that we add an abort rule. The challenger raises event $\text{abort}_{\text{eph}}$ and aborts, if during the simulation an ephemeral key (say $X$) replied by an oracle $\pi_s^\ast$ but it has been sample by another oracle or sent by adversary before. Since there are $\rho \ell$ such ephemeral keys
would be sampled uniform randomly from \(\{0, 1\}^\lambda\). Thus, the event \(\text{abort}_\text{eph}\) occurs with probability \(\Pr[\text{abort}_\text{eph}] \leq \frac{(\rho \ell)^2}{2^n}\). We have that

\[
\text{Adv}_0 \leq \text{Adv}_1 + \frac{(\rho \ell)^2}{2^n}.
\]

Note that the ephemeral key chosen by each oracle is unique in this game.

**Game 2.** In this game we want to make sure that there is no collision among the hash values of Diffie-Hellman keys. Technically, we add an abort condition, namely the challenger proceeds exactly as before, but raises event \(\text{abort}_\text{hash}\) and aborts if there exist two distinct (either ephemeral or long-term) public keys \(M\) and \(N\) such that \(\text{TCRHF}(M) = \text{TCRHF}(N)\). Obviously the \(\Pr[\text{abort}_\text{hash}] \leq \epsilon_{\text{TCRHF}}\), according to the security property of underlying hash function. Thus we have

\[
\text{Adv}_1 \leq \text{Adv}_2 + \epsilon_{\text{TCRHF}}.
\]

**Game 3.** This game proceeds as previous game, but \(C\) aborts if one of the following guesses fails: (i) the freshness case occurred to test oracle from the set \(\{C_1, C_9, C_{13}, C_{14}\}\), (ii) the test oracle \(\pi_{A_i}^*\), (iii) its partner parties \(\hat{B}\) and \(\hat{C}\), and (iv) corresponding oracles (if any) \(\pi_{D_i}^*\) (\(D \in \{\hat{B}, \hat{C}\}\)) such that \(\pi_{A_i}^*\) has a matching session to \(\pi_{D_i}^*\), in terms of specific guessed freshness case. Since there are four considered fresh cases, \(\ell\) parties and at most \(\rho\) oracles for each party, then the probability that all above guesses of \(C\) are correct is at least \(1/4(\rho \ell)^3\). Thus we have that

\[
\text{Adv}_2 \leq 4(\rho \ell)^3 \cdot \text{Adv}_3.
\]

**Game 4.** Please first note that there are at least three uncompromised (either long-term and ephemeral) Diffie-Hellman keys which are used by test oracle to generate its key material \(k^*\), as otherwise the test oracle is not g-eCK-fresh any more. We call such guessed three uncompromised DH keys as **target DH keys**.

Technically, this game is proceeded as previous game, but the challenger \(C\) replaces the key material \(k_i^*\) with random value \(\tilde{k}_i^*\) for oracles \(\{\pi_{s_i}^*: i \in [\ell], s \in [\rho]\}\) which satisfy the following conditions:

- The \(k_i^*\) is computed involving the three **target DH keys** which are guessed by \(C\) for test oracle, and
- These **target DH keys** used by \(\pi_{s_i}^*\) are from three distinct parties.

Furthermore, for two oracles satisfying above both conditions which have matching sessions, we could use the same modified key material to generate corresponding session
key. The second condition is necessary, because the adversary can easily result in one oracle receiving DH keys from certain party which are all uncompromised via e.g. Send query EstablishParty queries. On the other side, if those uncompromised DH keys are not from distinct parties, that might imply all DH keys from certain party are chosen (or revealed) by adversary. In this case, the adversary can compute the session key herself. The above two conditions are used to ensure that the changed key materials of oracles can not be trivially generated by adversary. This also enables us to embed CBDDH challenge instance into the simulation of those modified oracles.

If there exists an adversary \( \mathcal{A} \) can distinguish the Game 4 from Game 3 then we can use it to construct a distinguisher \( \mathcal{D} \) to solve the CBDDH problem as follows. Given a CBDDH challenge instance \( (g, g^a, \Gamma) \in G^2 \times G_T \), the goal of \( \mathcal{D} \) is to determine whether \( \Gamma = e(g, g)^{h^3} \) or a random element from \( G_T \) where \( g \) is a generator of \( G \). Let \( p(h) = p_0 + p_1 h + p_2 h^2 + p_3 h^3 \) be a polynomial of degree 3 over \( \mathbb{Z}_p^* \). The detail form of this polynomial will be discussed in the simulation based on specific freshness case. Let \( q(h) = q_0 + q_1 h + q_2 h^2 + q_3 h^3 \) be a random polynomial of degree 3 over \( \mathbb{Z}_p^* \). In the following, \( \mathcal{D} \) simulates the challenger for \( \mathcal{A} \) as previous game but with the some modifications based on its correct guesses (otherwise it aborts).

1. **Case C1.** In this case \( \mathcal{D} \) does the following modifications:

   a) Set \( X^* := g^{ur_x}, Y^* := g^{ur_y} \) and \( Z^* := g^{ur_z} \) where \( r_x, r_y, r_z \leftarrow \mathbb{Z}_p^* \).

   b) Compute the key material of test oracle and its partner oracles as:

   \[
   \begin{align*}
   k_A^* &= k_B^* = k_C^* := \Gamma^{ur_xr_yr_z}e(CZ^*, BY^*)^a \cdot e(Z^*, X^*)^b \cdot e(BY^*, X^*)^c. \\
   \end{align*}
   \]

   c) Compute those tags of test oracle and its partner oracles as:

   \[
   \begin{align*}
   t_X^* := (X^*)^{r_3(h X^*)^3 + r_2(h X^*)^2 + r_1 h X^* + r_0}, \\
   t_Y^* := (Y^*)^{r_3(h Y^*)^3 + r_2(h Y^*)^2 + r_1 h Y^* + r_0}, \\
   t_Z^* := (Z^*)^{r_3(h Z^*)^3 + r_2(h Z^*)^2 + r_1 h Z^* + r_0}.
   \end{align*}
   \]

2. **Case C9.** We assume there is an oracle \( \pi_B^* \) having matching session to test oracle without loss of generality. In this case \( \mathcal{D} \) does the following modifications:

   a) Set \( C := g^{ur_c}, X^* := g^{ur_x} \) and \( Y^* := g^{ur_y} \) where \( r_c, r_x, r_y \leftarrow \mathbb{Z}_p^* \).

   b) Set polynomial \( p(h) \) to satisfy that \( p(h) = (h - h X^*)(h - h Y^*)(h - h C) \), where

   \[
   h X^* = \text{TCRHF}(Y^*), h Y^* = \text{TCRHF}(X^*) \text{ and } h C = \text{TCRHF}(C).\]

   c) Set \( u_i = g^{ur_i}g^{r_i} \) for \( 0 \leq i \leq 3 \).

   d) Compute the tags \( t_C = C^{q(hc)}, t_Y^* = (Y^*)^{q(h Y^*)} \) and \( t_X^* = (Z^*)^{q(h Y^*)} \).

   e) Compute the key material of test oracle and its partner oracle as:

   \[
   \begin{align*}
   k_A^* &= k_B^* := \Gamma^{ur_xr_yr_z}e(CV^*, BY^*)^a \cdot e((\frac{r_y}{q(h Y^*)})^{r_x/p(h Y^*)}, BY^*) \cdot e(C, X^*)^b. \\
   \end{align*}
   \]

f) Compute the key material of other oracles of \( \mathcal{C} \) in terms of the following situations:
There are at most two DH keys in \(\{F, V, D, W\}\) which are equivalent to keys in the set \(\{X^*, Y^*\}\) and from different parties. Since the adversary can either register these keys as public key for dishonest users or replay them as ephemeral key. Recall that these DH keys in \(\{F, V, D, W, C, Z\}\) should be distinct in corresponding sid. We assume that \(E = X^*\) and \(D = Y^*\) for example, then the \(k^1_C\) is computed as: 
\[
\begin{align*}
\hat{t}^C := \Gamma_{r_a r_b r_c} e((t_w^{(h)} W^r_{(h^W)}), FV) \cdot e((t_v^{h(l)})^{r_c/p(h^V)}, D).
\end{align*}
\]

- The DH keys (either long-term or ephemeral public key) from one party do not belong to the set \(\{X^*, Y^*\}\). We assume that \(\{F, V\} \notin \{X^*, Y^*\}\) for example, then the \(k^1_C\) is computed as:
\[
\begin{align*}
\hat{t}^C := e((t_k^{E(h)})^{r_c/p(h^E)} (t_w^{h(l)})^{r_c/p(h^W)}, DW) \cdot e(FV, DW)^z
\end{align*}
\]
when \(V \notin \{X^*, Y^*\}\).

3. Case C13. In this case \(D\) does the following modifications:

a) Set \(A := g^{\mu r_a}, B := g^{\mu r_b}\), and \(C := g^{\mu r_c}\), where \(r_a, r_b, r_c \leftarrow \mathbb{Z}_p\).

b) Set polynomial \(p(h)\) to satisfy that \(p(h) = (h - h_A)(h - h_B)(h - h_C)\), where \(h_A = TCRHF(A), h_B = TCRHF(B)\) and \(h_C = TCRHF(C)\).

c) Set \(u_i = g^{\mu p_i} g^{h_i}\) for \(0 \leq i \leq 3\). Please note that we have that \(u_0 u_1^h u_2^h u_3^h = g^{\mu p(h)} g^{h(h)}\) and \(u_0 u_1^h u_2^h u_3^h = g^{\mu p(h)} g^{h(h)}\).

d) Compute the tags \(t_A = A^{q(h_A)}, t_B = B^{q(h_B)}\) and \(t_C = C^{q(h_C)}\).

e) Replace the value \(e(B, C)^x\) with \(\Gamma_{r_a r_b r_c}^x\) when compute the key material \(k\) of oracles \(\pi_A^x, \pi_B^x, \pi_C^x\) which involve all public keys \((A, B, C)\) including the test oracle \(\pi_A^x\), more specifically:

\[
\begin{align*}
\hat{t}^A := \Gamma_{r_a r_b r_c} e(CV, BW)^x \cdot e((t_w^{(h)} W^r_{(h^W)}), C).
\end{align*}
\]

\[
\begin{align*}
\hat{t}^B := \Gamma_{r_a r_b r_c} e(CV, AW)^y \cdot e((t_w^{h(l)})^{r_c/p(h^V)}, C).
\end{align*}
\]

\[
\begin{align*}
\hat{t}^C := \Gamma_{r_a r_b r_c} e(BV, AW)^z \cdot e((t_w^{h(l)})^{r_c/p(h^W)}, B).
\end{align*}
\]

f) Compute the secret key material \(k\) for other oracles of parties \(\hat{A}, \hat{B}\) and \(\hat{C}\), following the similar approach as did in the proof of Case C9 when computing the key material for oracles of uncorrupted party \(\hat{C}\) (i.e. the modification in the step 2f). The common point here is that we should simulate the key material for situations when there exist DH keys equal to challenged DH keys (i.e. \(A\) or \(B\) or \(C\)).

4. Case C14. In this case \(D\) does the following modifications:

a) Set \(X^* := g^{\mu r_x}, B := g^{\mu r_b}\) and \(C := g^{\mu r_c}\) where \(r_x, r_b, r_c \leftarrow \mathbb{Z}_p\).
b) Set polynomial \( p(h) \) to satisfy that \( p(h) = (h - h_X^*)(h - h_B)(h - h_C) \), where \( h_B = TCRHF(B) \), \( h_X^* = TCRHF(X^*) \) and \( h_C = TCRHF(C) \).

c) Set \( u_i = g^{p_i}g^{q_i} \) for \( 0 \leq i \leq 3 \).

d) Compute the tag \( t_B = B^q(h_B) \), \( t_C = C^q(h_C) \) and the tag \( t_X^* = (X^*)^q(h_X^*) \).

e) Compute the key material \( k_A^* \) of test oracle \( \pi_A^* \), \( k_B^l \) of oracles \( \pi_B^l \) and \( k_C^l \) of oracles \( \pi_C^l \) which compute the session keys using public keys \( (X^*, B, C) \) as:

- \( k_A^* := \Gamma^{r_xv_v}e(CV, BW)^{a \cdot e((\frac{t_w}{W^q(h_W)}))^{r_x/p(h_W)}, C) \).

- \( k_B^l := \Gamma^{r_xv_v}e(CV, X^* D)^{y \cdot e((\frac{t_D}{D^q(h_D)}))^{r_x/p(h_D)}, C) \).

- \( k_C^l := \Gamma^{r_xv_v}e(BW, X^* D)^{z \cdot e((\frac{t_D}{D^q(h_D)}))^{r_x/p(h_D)}, B) \).

f) Change the computation of secret key material \( k \) of other oracles of \( \hat{B} \) and \( \hat{C} \) following the similar approach as did in the proof of Case C9 when computing the key material for oracles of uncorrupted party \( C \). One could think of replacing the symbols (e.g. \( Y^* \)) in the step 2f with the symbols (e.g. \( B \)) in this case.

Those modified tags are consistent with the original form. We make use of the fact there is no collision on those hash values due to the result of previous game. To answer the \texttt{RevealKey} query for those modified oracles, the \( D \) will use the changed key material (e.g. \( k_B^l \)) to compute the final session key as protocol specification. With respect to the other queries, the \( D \) simulates them honestly as the challenger using corresponding values chosen by herself. Without flipping the bit \( b \), the \texttt{Test}-query is replied with the session key which is computed using modified key material. Based on the condition that all guesses of \( D \) are correct, if \( \Gamma = e(g, g)^{\alpha_3} \), then the simulation is equivalent to Game 3; otherwise the simulation is equivalent to Game 4. At the end of the game, \( D \) returns what \( A \) returns to the \texttt{CBDDH} challenger. If \( A \) can distinguish the real key from the random value, that implies \( D \) solves the \texttt{CBDDH} problem. We therefore obtain that

\[
\text{Adv}_3 \leq \text{Adv}_4 + \epsilon_{\text{CBDDH}}.
\]

\textbf{Game 5.} In this game, we change function \( \text{PRF}(k_A^*, \cdot) \) to a truly random function for test oracle and its partner oracles (if they exist). We make use of the fact that the secret seed \( k_A^* \) of test oracle is a truly random value. Any PPT algorithm distinguishing the Game 5 from Game 4 implies that it is able to break the security of the pseudo-random function \( \text{PRF} \). Thus we have that

\[
\text{Adv}_4 \leq \text{Adv}_5 + \epsilon_{\text{PRF}}.
\]
Note that in this game the session key returned by Test-query is totally a truly random value which is independent to the bit \( b \) and any messages. Thus the advantage that the adversary wins this game is \( \text{Adv}_5 = 0 \).

Sum up the probabilities from Game 0 to Game 5, we proved this theorem.

\[ \square \]

### 6.3 A GAKE Construction from Multilinear Maps

An interesting work is to extend the proposed 3AKE scheme to GAKE scheme with more than three group members. Based on bilinear groups might be impossible to achieve so. Since we can not get an aggregate long-term shared key for a group of members from bilinear map. However, Boneh and Silverberg [BS02] have given us inspiration on how to generalize the 3AKE to GAKE by exploiting multilinear maps.

#### 6.3.1 Protocol Description

**Setup:** The proposed protocol takes as input the following building blocks which are initialized respectively in terms of the security parameter \( \kappa \in \mathbb{N} \) and upper-bound of group size \( n + 1 \):

- \( n \)-multilinear groups \( \mathcal{MLG} = (\mathcal{G}, \mathcal{G}_T, g, p, me) \xrightarrow{\$} \text{MLG.Gen}(\kappa, n) \) and a set of random values \( \{u_j\}_{0 \leq j \leq n+1} \xrightarrow{\$} \mathcal{G} \).

- a target collision resistant hash function \( \text{TCRHF}(h_{\text{TCRHF}}, \cdot) : \mathcal{K}_{\text{TCRHF}} \times \mathcal{G} \rightarrow \mathbb{Z}_p \), where \( h_{\text{TCRHF}} \xrightarrow{\$} \text{TCRHF.KGen}(1^\kappa) \), and

- a pseudo-random function family \( \text{PRF}(\cdot, \cdot) : \mathcal{G}_T \times \mathcal{D}_{\text{PRF}} \rightarrow \mathcal{K}_{\text{AKE}} \).

Let \( pms := (\mathcal{MLG}, \{u_j\}_{0 \leq j \leq n+1}, h_{\text{TCRHF}}) \) be the variable used to store the public system parameters.

**Long-term Key Generation and Registration:** On input \( pms := (\mathcal{MLG}, \{u_j\}_{0 \leq j \leq n+1}, h_{\text{TCRHF}}) \), a party \( \hat{A} \) may run an efficient key generation algorithm \( (sk_{\hat{D}}, pk_{\hat{D}}, \emptyset) \xrightarrow{\$} \text{ORGAKE.KGen}(pms, \hat{D}) \) to generate the long-term key pair for a party \( \hat{D} \) as: \( sk_{\hat{D}} = d \xrightarrow{\$} \mathbb{Z}_p^*, pk_{\hat{D}} = (D, t_D) \), where \( D = g^a, t_D := \prod_{j=0}^{n+1} u_j^{h_{\hat{D}}^j} \) and \( h_A = \text{TCRHF}(A) \). Please note that we allow arbitrary key registration, i.e. the adversary is able to query

EstablishParty(\( \hat{D}, pk_{\hat{D}}, \emptyset \)) with \( pf_{\hat{D}} = \emptyset \).

Let \( \omega \) denote the size of group for a protocol instance such that \( 2 \leq \omega \leq n + 1 \). An important attribute for a GAKE protocol is the scalable group size. In the following we show our construction for protocol execution phase which is scalable with range
between 2 and $n + 1$. Recall that the upper bound of group size is determined by the n-multilinear map.

**Protocol Execution:** We consider the protocol execution for a protocol instance with $\omega$ group members denoted by $(\tilde{D}_1, \tilde{D}_2, \ldots, \tilde{D}_\omega)$, where each party $\tilde{D}_i$ ($1 \leq i \leq \omega$) has long-term key $D_i$. In the key exchange phase, each party $\tilde{D}_i$ generates an ephemeral key $X_i = g^{x_i}$, computes tag $t_{X_i} = \prod_{j=0}^{n+1} u_j^{h_{X_i}}$ and broadcasts $(\tilde{D}_i, X_i, t_{X_i})$ to its intended communication partners, where $x_i \in \mathbb{Z}_p^*$ and $h_{X_i} := TCRHF(X_i)$. Upon receiving all messages $\{\tilde{D}_i, X_i, t_{X_i}\}_{1 \leq i \leq \omega}$ from each session participant, the party $\tilde{D}_i$ rejects the session if the consistency check on one of the received either long-term or ephemeral keys fails, i.e. $me(t_{W_l}, g, \ldots, g) \neq me(\prod_{j=0}^{n+1} u_j^{h_{W_l}}, W_l, g, \ldots, g)$ where $W_l \in \{D_l, X_l\}$ for $1 \leq l \leq \omega, l \neq i$ and $h_{W_l} = TCRHF(W_l)$. The party $\tilde{D}_i$ sets $\text{sid} := \tilde{D}_i||D_i||t_{D_i}||X_i||t_{X_i}||\ldots||D_\omega||t_{D_\omega}||X_\omega||t_{X_\omega}$, and rejects the session if some values recorded in $\text{sid}$ are identical. To this end, the party $\tilde{D}_i$ generates the key material $k := me(D_1X_1, \ldots, D_{i-1}X_{i-1}, D_{i+1}X_{i+1}, \ldots, D_\omega X_\omega, D_\omega X_\omega)^{d_i+x_i}$ and session key $k_e := \text{PRF}(k, \text{sid})$, where the values $D_0, X_0, D_{\omega+1}, X_{\omega+1}$ are ‘empty’ which should be omitted. Other parties in this group will do the similar procedures to generate the session key.

Please note that the scalability is achieved generally by setting all Diffie-Hellman keys after the position $\omega$ in n-multilinear map $me$ to be $D_\omega X_\omega$. This is possible since at least one DH key in $(D_\omega, X_\omega)$ is not compromised by adversary in the security game. As otherwise such session is no longer fresh in terms of Definition 28.

**Implementation and Session States:** We assume that the maximum states of party $\tilde{D}_i$ allowing for leakage from a session consist of ephemeral private key $x_i$ – namely those values would be stored in the variable in the state variable $st$ of each oracle at any time. The implementation scenario is similar to the three party case presented in Section 6.2, namely generate the $k$ and $k_e$ on secure device.

**Remark 2.** The above construction implies the proposed tripartite AKE protocol in Section 6.2 if the parameter of n-multilinear map such that $n = 2$ which is equivalent to bilinear map. Then the scalable construction of GAKE could also yield a two party eCK secure AKE protocol that might be of independent interesting. It is not hard to see that the security of such two party AKE protocol can be proved without random oracles based on CBDDH assumption in the g-eCK model when group size equals two (i.e. then it implies the eCK model).

---

As for multilinear maps which are implemented with a series of bilinear maps, e.g. the framework by Garg et al. [GGH13, GGH12], one could use the bilinear map $e$ instead of $me$ in those consistency check operations for efficiency consideration. One could also use the similar technique in Section 5.3.2 to merge those consistency checks.
6.3.2 Security Analysis

We show the security of above group AKE protocol in the g-eCK model.

**Theorem 7.** Assume each ephemeral key chosen during key exchange has bit-size \( \lambda \in \mathbb{N} \). Suppose that the nMDDH problem is \((t, \epsilon_{nMDDH})\)-hard in the symmetric multilinear groups \( \mathcal{MLG} \), the TCRHF is \((t, \epsilon_{TCRHF})\)-secure target collision resistant hash function family, and the PRF is \((q_{prf}, t, \epsilon_{PRF})\)-secure pseudo-random function family. Then the proposed protocol of size \( 2 \leq \omega \leq n + 1 \) is \((t', \epsilon)\)-g-eCK-secure in the sense of Definition 29 with \( t' \approx t \), \( q_{prf} \geq n + 1 \) and

\[
\epsilon \leq \frac{(\rho \ell)^2}{2^\lambda} + \epsilon_{TCRHF} + (n + 2)(\rho)^{n+1}\left(\frac{\ell}{n+1}\right) \cdot (\epsilon_{nMDDH} + \epsilon_{PRF}).
\]

**Proof.** Basically, the proof can be generalized from the proof of Theorem 6 due to the intimate relationship between proposed 3AKE and GAKE schemes. We will focus on the largest group size \( n + 1 \) without loss of generality, because we need to evaluate the maximum advantage of adversary to break the protocol. For the test query involving group of size smaller than \( n + 1 \), the simulation is quite similar.

Let \( S_\delta \) be the event that the adversary wins the security experiment in Game \( \delta \). Let \( \text{Adv}_\delta := \Pr[S_\delta] - 1/2 \) denote the advantage of \( A \) in Game \( \delta \).

**Game 0.** This is the original game with adversary \( A \). The system parameters are chosen honestly by challenger as protocol specification. However, the challenger chooses \( n + 1 \) uniform random values \( \{r_j\}_{0 \leq j \leq n} \), and sets \( u_j := g^{r_j} \) as public parameters. Thus we have that

\[
\Pr[S_0] = 1/2 + \epsilon = 1/2 + \text{Adv}_0.
\]

**Game 1.** This game proceeds as the same as the Game 2 in the proof of Theorem 6. With the similar argument from the proof of Game 2 of Theorem 6, we have that

\[
\text{Adv}_0 \leq \text{Adv}_1 + \frac{(\rho \ell)^2}{2^\lambda} + \epsilon_{TCRHF}.
\]

**Game 2.** This game proceeds as previous game, but \( C \) aborts if one of the following guesses fails: (i) the freshness case occurred to test oracle from all \( n + 2 \) possibilities, (ii) the test oracle, (iii) the \( n \) intended communication partners of test oracle, and (iv) all oracles (if they exist in terms of specific guessed freshness case) which have matching session to test oracle. Since there are \( n + 2 \) fresh cases that need to do proof simulation, \( \ell \) parties at all and at most \( \rho \) oracles for each party, then the probability that all above guesses of \( C \) are correct is at least \( \frac{1}{(n+2)(\rho)^{n+1}\binom{n+1}{\ell}} \). Thus we have that
\[ \text{Adv}_1 \leq (n + 2)(\rho)^{n+1} \left( \frac{\ell}{n+1} \right) \cdot \text{Adv}_2. \]

We lose a factor \((n + 2)(\rho)^{n+1} \left( \frac{\ell}{n+1} \right)\) here which is exponential in group size \(n\). Hence, in order to make the overall advantage of adversary to be negligible, one may need to use a larger security parameter or to limit the maximum group members.

**Game 3.** Please note that the g-eCK freshness definition guarantees that for our protocol there are at least \(n + 1\) Diffie-Hellman (DH) keys from all session participants of test fresh oracle are not compromised by adversary. We call such guessed \(n + 1\) uncompromised DH keys as **target DH keys**. This game is proceeded as previous game, but the challenger \(C\) replaces the key material \(k^s_i\) with random value \(\tilde{k^s_i}\) for oracles \(\{\pi^s_i : i \in [\ell], s \in [\rho]\}\) which satisfy the following conditions:

- The \(k^s_i\) is computed involving the \(n + 1\) target DH keys which are guessed by \(C\) for test oracle, and
- Those target DH keys used by \(\pi^s_i\) are from \(n + 1\) distinct parties.

Of course if two oracles have matching sessions and satisfy both above conditions, then we could use the same modified random key material to generate corresponding session key. The above two conditions ensure that the changed key materials of oracles can not be trivially generated by adversary. This also enables us to embed \(nMDDH\) challenge instance into the simulation of all oracles satisfying above conditions. The second condition is used to exclude the situation that the DH keys from some party are all compromised in which case the adversary can simply compute the session key. If there exists an adversary \(A\) can distinguish the Game 3 and 2 then we can use it to construct a distinguisher \(D\) to solve the \(nMDDH\) problem. Given a \(nMDDH\) challenge instance \((g, g^\mu, \Gamma) \in G^2 \times G_T\), the goal of \(D\) is to determine whether \(\Gamma = me(g, \ldots, g)^{h^{n+1}}\) or a random element from \(G_T\) where \(g\) is a generator of \(G\). Meanwhile, \(D\) simulates the challenger for \(A\) as previous game but with the following modifications based on its correct guesses (otherwise it aborts). We highlight that, after all those correct guesses, \(D\) knows the facts about which parties’ long-term keys are not corrupted (if any) and which oracles’ ephemeral keys are not revealed (if any), under specific guessed freshness cases. Let \(p(h) = \sum_{j=0}^{n+1} h^j = (h - h_{W_1}) \ldots (h - h_{W_{n+1}})\) be a polynomial of degree \(n + 1\) over \(\mathbb{Z}_p^*\) such that \(p(h_{W_1}) = p(h_{W_2}), \ldots, p(h_{W_{n+1}}) = 0\) where \(h_{W_j} = TCRHF(W_j)\) for \(1 \leq j \leq n + 1\) and each \(W_j\) is either uncorrupted long-term key \(D_j\) or uncompromised ephemeral key \(X_j\) in specific freshness case. Let \(q(h) = \sum_{j=0}^{n+1} q^j h^j\) be a random polynomial of degree \(n + 1\) over \(\mathbb{Z}_p^*\). It will also set \(u_j = g^{\mu q_j} g^j\) for \(0 \leq j \leq n + 1\). Meanwhile, we would plug the challenge value \(g^\mu\) to all
n + 1 target uncompromised DH keys in specific (guessed) freshness case, i.e. \( D \) generates the DH key as \( W_j = g^{w_j} \), where \( w_j \leftarrow Z_p^* \). Moreover, the tag \( t_{W_j} \) of \( W_j \) would be computed as \( t_{W_j} = W_j^{q(h_{W_j})} \). The remaining problem is to simulate the \texttt{RevealKey} query and \texttt{Test} query correctly in terms of freshness case.

On the next we discuss how to simulate the key material for any oracle \( \pi_i^s (i \in [\ell], s \in [\rho]) \), including test oracle and its partner oracle (if they exists). In the sequel, we let \((D_1,t_{D_1},X_1,t_{X_1})\) denote the values generated for oracle \( \pi_1^s \), and let \( \{D_j,t_{D_j},X_j,t_{X_j}\}_{2 \leq j \leq n+1} \) denote a set of values received by oracle \( \pi_i^s \) from its intended communication partners.\(^4\) We consider the following cases (which cover all) concerning the DH keys of \( \pi_i^s \):

1. Case 1: the ephemeral key \( X_1 \) is generated from challenge value \( g^\mu \).
2. Case 2: the long-term key \( D_1 \) is generated from challenge value \( g^\mu \).
3. Case 3: neither long-term key \( D_1 \) nor ephemeral key \( X_1 \) is generated from challenge value \( g^\mu \).

It is not hard to see, in the Case 3 \( D \) can simulate the key honestly as protocol specification. Thus we only need to do modifications on oracles \( \pi_i^s \) under Case 1 and Case 2. With respect to the Case 1, the \( d_1 \leftarrow Z_p^* \) is chosen by \( D \) as protocol specification and \( X_1 \) is generated using challenge value as \( X_1 := g^{\mu r_1} \), where \( r_1 \leftarrow Z_p^* \). Then the tag \( t_{X_1} \) can be computed as \( t_{X_1} := X_1^{q(h_{X_1})} \) and \( h_{X_1} = \text{TCRHF}(X_1) \). With respect to the Case 2, the \( x_1 \leftarrow Z_p^* \) might be chosen by \( D \) and \( D_1 \) can be set as \( D_1 := g^{\mu r_1} \), where \( r_1 \leftarrow Z_p^* \). The tag \( t_{D_1} \) can be computed as \( t_{D_1} := D_1^{q(h_{D_1})} \) and \( h_{D_1} = \text{TCRHF}(D_1) \).

Let \( W_1 \) denote the DH key generated for oracle \( \pi_1^s \) such that \( W_1 \in \{D_1,X_1\} \) and \( W_1 \) is generated using challenge value as \( g^{\mu r_1} \), where \( r_1 \in \{r_{X_1},r_{D_1}\} \) depending on the value of \( W_1 \). We further let \( \overline{W}_1 = g^{\overline{w}_1} \) denote the DH key generated for oracle \( \pi_1^s \) such that \( \overline{W}_1 \in \{D_1,X_1\} \) and \( \overline{W}_1 \) is not generated using challenge value. Then we could rewrite the key material \( k_i^s \) of oracle \( \pi_i^s \) as

\[
k_i^s = \text{me}(D_2X_2, \ldots, D_{n+1}X_{n+1})^{d_1+x_1} = \text{me}(D_2X_2, \ldots, D_{n+1}X_{n+1})^{\overline{w}_1+w_1} = \text{me}(D_2X_2, \ldots, D_{n+1}X_{n+1})^{\overline{w}_1} \cdot \text{me}(D_2X_2, \ldots, D_{n+1}X_{n+1})^\overline{w}_1.
\]

Since the \( \overline{w}_1 \) is chosen by \( D \) then it is able to compute the value

\[
\alpha = \text{me}(D_2X_2, \ldots, D_{n+1}X_{n+1})^\overline{w}_1.
\]

For both above cases, we further consider the following disjoint event that covers all possibilities.

\(^4\)Please forget the (subscripts) positions of DH keys recorded in \texttt{sid}^i of \( \pi_i^s \) for the time being. We here need to differentiate the DH keys generated for oracle \( \pi_i^s \) with other DH keys received by \( \pi_i^s \) in the following modification, even though those DH keys of \( \pi_i^s \) might be located in different position in \texttt{sid}^i rather than the first place.

118
Event 1: Firstly, we consider the event that every DH key tuple \((D_j, X_j)\) for \(2 \leq j \leq n + 1\) received by oracle \(\pi_i^s\) consists of one DH key that is computed using challenge value \(g^\mu\). As all values recorded in \(\text{id}_i^s\) are distinct, so that in each received DH key tuple \((D_j, X_j)\) there is at most one DH key that is generated using challenge value in this event. We further let \(W_j = g^{\mu r_j}\) for \(2 \leq j \leq n + 1\) denote the DH key received by oracle \(\pi_i^s\) such that \(W_j \in \{D_j, X_j\}\) and \(W_j\) is generated using challenge value. And we let \(\overline{W}_j = g^{\beta'}\) for \(2 \leq j \leq n + 1\) denote the DH key received by oracle \(\pi_i^s\) such that \(\overline{W}_j \in \{D_j, X_j\}\) and \(\overline{W}_j\) is not generated using challenge value for \(2 \leq j \leq n + 1\). Then in this event, \(\mathcal{D}\) could compute the key material \(k_i^s\) using the value \(\Gamma\), randomness \(r_{w_i}\) and the value \(g^{\mu r_j}\) extracted from \(t_{\overline{W}_j}\), and \(n\)-multilinear map operations. To elaborate the simulation of \(k_i^s\), we rewrite the \(\beta := me(D_2 W_2, \ldots, D_{n+1} X_{n+1})^{w_1}\) as following:

\[
\beta := me(g^{\mu_2 r_1}, D_3 X_3, \ldots, D_{n+1} X_{n+1}) \cdot me(W_2, D_3 X_3, \ldots, D_{n+1} X_{n+1})^{w_1} \\
= me(g^{\mu_2 r_1}, D_3 X_3, \ldots, D_{n+1} X_{n+1}) \cdot me(W_2, g^{\mu_3 r_1}, D_4 X_4, \ldots, D_{n+1} X_{n+1})^{w_1} \\
\quad \cdot me(W_2, W_3, D_4 X_4, \ldots, D_{n+1} X_{n+1})^{w_1} \\
= me(g^{\mu_3 r_1}, D_3 X_3, \ldots, D_{n+1} X_{n+1}) \cdot me(W_2, g^{\mu_4 r_1}, D_4 X_4, \ldots, D_{n+1} X_{n+1})^{w_1} \\
\quad \cdot me(W_2, W_3, g^{\mu_4 r_1}, D_5 X_5, \ldots, D_{n+1} X_{n+1}) \\
\quad \vdots \\
\quad \cdot me(W_2, W_3, W_4, \ldots, W_{n-1}, g^{\mu_n r_1}, D_{n+1} X_{n+1})^{w_1} \\
\quad \cdot me(W_2, W_3, W_4, \ldots, W_{n-1}, W_n, g^{\mu_n r_{w_n}}) \cdot me(W_2, W_3, \ldots, W_{n+1})^{w_1}.
\]

The above ‘expansion’ of the equation is only conceptual that is consistent to the original computation of \(\beta\). However this enables us to embed the challenge value \(\Gamma\) into the key material \(k_i^s\) without knowing \(w_1\). More specifically we change \(\beta\) to \(\beta'\) by replacing the value \(me(W_2, W_3, \ldots, W_{n+1})^{w_1}\) in above computation of \(\beta\) with value \(\Gamma^{r_{w_j} - r_{w_{n+1}}}\) and computing values \(g^{\mu r_j}\) from tag \(t_{\overline{W}_j}\) as \(g^{\mu r_j} = \left(\frac{\overline{W}_j}{\text{id}_i^s\overline{W}_j}\right)^{\frac{1}{\mu r_j}}\) where \(t_{\overline{W}_j} \in \{t_{D_j}, t_{X_j}\}\) and \(2 \leq j \leq n + 1\).

Eventually we compute the key material \(k_i^s = \alpha \cdot \beta'\) and use it to compute the final session key of oracle \(\pi_i^s\).

Event 2: On the second, we consider the event that there exists one DH key tuple \((D_j, X_j)\) (\(2 \leq j \leq n + 1\)) received by oracle \(\pi_i^s\) which are all not generated using challenge value \(g^\mu\). Then, in order to simulate the key material \(k_i^s\), the jobs of \(\mathcal{D}\) are only to compute \(g^{\mu d_j}\) from \(t_{D_j}\) (if \(D_j\) is chosen by adversary, as otherwise \(\mathcal{D}\) knows corresponding exponent \(d_j\)) as \(g^{\mu d_j} := \left(\frac{t_{D_j}}{g^{\mu d_j}}\right)^{\frac{1}{\mu d_j}}\) and to compute \(g^{\mu x_j}\) from \(t_{X_j}\) as \(g^{\mu x_j} := \left(\frac{t_{X_j}}{X_j^2}\right)^{\frac{1}{\mu X_j}}\). Let \(\{\eta_l\}\) for \(1 \leq l \leq n - 1\) be a set of variables each of which stores distinct integer number ranging
from 2 to \( n + 1 \) except for \( j \). Thus the key material is generated as 
\[
  k_i^s = \alpha \cdot me(g^{\mu d_j r_{w1}} g^{\mu x_j r_{w1}}, D_{\eta_1} X_{\eta_1}, \ldots, D_{\eta_{n-1}} X_{\eta_{n-1}}),
\]
which is consistent to original form.

In a nutshell, \( \mathcal{D} \) is able to simulate all session keys appropriately in terms of the tags of both ephemeral key and long-term key. If \( \Gamma = me(g, \ldots, g)^{\mu n+1} \) then the simulation is exactly equivalent to previous game, otherwise it equals to this game. By applying the security of \( nMDDH \) assumption, we therefore obtain that

\[
  \text{Adv}_2 \leq \text{Adv}_3 + \epsilon_{nMDDH}.
\]

**Game 4.** In this game, we change function \( \text{PRF}(\tilde{k}^*_i, \cdot) \) to a truly random function for test oracle and its partner oracles (if they exist). We make use of the fact, that the secret seed \( \tilde{k}^*_i \) of test oracle is a truly random value. If there exists a polynomial time adversary \( \mathcal{A} \) can distinguish the Game 4 from Game 3. Then we can construct an algorithm \( \mathcal{B} \) using \( \mathcal{A} \) to break the security of \( \text{PRF} \). Exploiting the security of \( \text{PRF} \), we have that

\[
  \text{Adv}_3 \leq \text{Adv}_4 + \epsilon_{\text{PRF}}.
\]

Note that in this game the session key returned by \text{Test}-query is totally a truly random value which is independent to the bit \( b \) and any messages. Thus the advantage that the adversary wins this game is \( \text{Adv}_4 = 0 \).

Sum up the probabilities from Game 0 to Game 4, we proved this theorem. \( \square \)
7 Conclusion and Future Work

We have shown several AKE constructions which are provably secure in different strong security models without random oracles. In this chapter we conclude the thesis via reviewing our general contributions and specify a number of interesting future works in conjunction with specific class of AKE protocol.

**AKE compiler.** We presented three compilers for building AKE protocols from passively secure KE protocols. Our compilers are secure in a very strong modified CK security model that allows for state reveals, PKI-related attacks and KCI 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 to be applicable to existing systems without requiring any modification. We raised an attention on a new theoretical RAP attack that needs to take care when defining session identifier for analyzed protocol in certain strong security model, e.g. mCK model or eCK model. We pointed out this attack was overlooked by a lot of previous works that might trivially invalidate their security results. Some solutions were discussed on how to avoid this attack in general ways. As for future work, it might be interesting to extend our results to group case, for instance to design compiler for constructing provably secure group AKE protocols from passively secure group KE protocols without any modifications. It is also an open question to formally prove the resilience of CIDPK attacks property for our compilers.

**One-round AKE Protocols.** We showed a generic construction GC-KKN for eCK-secure one-round two party AKE protocols in the standard model without NAXOS trick, which can be instantiated with passively secure KE scheme, IND-CCA secure KEM schemes and CKS-light secure NIKE schemes. In contrast to previous works in the standard model, the major merit of GC-KKN is its outstanding efficiency on implementing with secure device. In other word, only a small part of algorithms of GC-KKN need to be run on secure device to achieve state leakage resistant. We also gave a concrete protocol P1 based on the construction idea of GC-KKN. Similarly, P1 is motivated to further improve the efficiency on secure device wherein only one regular exponential operation is required. The P1 was proved eCK secure in the standard model under only standard assumptions including BDDH assumption and secure target collision resistant hash function family and pseudo-random function family. To our best
of knowledge, P1 is the first such protocol without NAXOS trick that can run under post-specified peer setting without knowing any information of communication peer at session activation.

In the later, we presented a new one-round 3AKE protocol based on bilinear maps that is provably secure in the g-eCK model. It is the first such scheme achieving this level of security in the standard model. We also made the first attempt to build a g-CK secure protocol with large scalable group size (more than three members). A candidate construction for such GAKE was given based on multilinear maps. Lots of future works along the line of one-round GAKE studied in this thesis could be done. As our GAKE solutions only provide weak perfect forward secrecy. An open question is to build one-round GAKE with perfect forward secrecy. One may borrow the idea from the compilers regarding how to achieve perfect forward secrecy for any existing secure one-round two party protocols with weak forward secrecy, e.g. compiler-MAC [BG11] and compiler-SIG [CF12]. The security reduction to \( \text{nMDDH} \) hard problem is exponential in \( n \). This would result in either usage of larger security parameter or limitation of the maximum group size. To seek a security proof with better security bound could be a research direction for future work. It might be also interesting to construct g-eCK secure GAKE protocol without multilinear maps.
Bibliography

Key Infrastructure Certificate Management Protocol (CMP). RFC 4210 
(Proposed Standard), September 2005.

key agreement protocols from pairings. In Kenneth G. Paterson, editor, 9th 
IMA International Conference on Cryptography and Coding, volume 2898 
of Lecture Notes in Computer Science, pages 332–359, Cirencester, UK, 

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstan-
tiable random-oracle-model scheme for a hybrid-encryption problem. In 
Christian Cachin and Jan Camenisch, editors, Advances in Cryptology – 
EUROCRYPT 2004, volume 3027 of Lecture Notes in Computer Science, 
Germany.

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach 
to the design and analysis of authentication and key exchange protocols 
(extended abstract). In 30th Annual ACM Symposium on Theory of Com-

Efficient one-round key exchange in the standard model. In Yi Mu, Willy 
Susilo, and Jennifer Seberry, editors, ACISP 08: 13th Australasian Confer-
ce on Information Security and Privacy, volume 5107 of Lecture Notes in 
Springer, Berlin, Germany.

[BCP01] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably 
authenticated group Diffie-Hellman key exchange – the dynamic case. In 
2248 of Lecture Notes in Computer Science, pages 290–309, Gold Coast, 


Bibliography


Bibliography


Bibliography


Zheng Yang

Curriculum Vitae

Personal Information

Date of birth: September 18th, 1982
Place of birth: ChongQing, China
Family status: Married
Child: Qiaoyi Yang (born 2011)
Languages: Mandarin (Mother tongue), English

Education

Oct 2009 – Present: PhD Study, Chair for Network- and DataSecurity, Faculty of Electrical Engineering and Information Technology, Ruhr-University Bochum, Germany. Supervisor: Prof. Dr. Jörg Schwenk. Anticipated graduation: Autumn 2013

Sep 2006 – Jun 2009: Master of Science in Engineering, Faculty of Computer Science, Chongqing University, China. Major in Computer Software And Theory

Sep 2001 – Jul 2005: Bachelor of Science in Engineering, Faculty of Computer Science and Technology, Chongqing Jiaotong University, China.

Computer Skills

Program Languages: JAVA, C, C++, SQL, PHP, PERL
Platforms: Linux, Windows Server

Working Experience


Publications


**Interests and Hobbies**

I love any sport but I prefer Football and Table Tennis. I love Trekking and Fishing on vocation.