Towards Strong Adaptive Corruption Security of Authenticated Key Exchange

Zheng Yang
Ruhr University Bochum,Germany
Zheng.Yang@rub.de

Ruhr University Bochum,
D-44801 Bochum, Germany

Abstract. In this paper we study strong adaptive corruption security definitions for authenticated key exchange (AKE) protocols. Many recent protocols for Authenticated Key Exchange have been proven correct in the CK01 or eCK security model. The new model is suggested to be at least as strong as previous models for authenticated key exchange protocols. However, we observe that there are several kinds of attacks on existing AKE protocols that beyond the current class of security definitions which further reveal the shortcomings in security proofs in related AKE security models, in particular concerning the protocols under eCK model. Since the two models are not formally comparable, we discuss the ambiguities of existing security definitions and then provide a general framework for defining AKE security when involve strong adversary capabilities. In which we formulate the timing of the authentication, key generation and key confirmation, for different classes of AKE protocols. In addition, we propose a new two-pass AKE protocol called Σ^y as an instance, which is proven secure in our proposed strong security definitions, under random oracle model and GDH assumption. In this protocol we show that our the proposed model, would also be a helpful guidance to design a secure protocol under strong adversary model. The intuition is generic: we embed the global unique identifier for unique-pairwise matching sessions into the key materials, before submitting to final key deviation function.

Keywords: Security model, Authenticated Key Exchange, SessionState, Ephemeral Key, Key Compromise Impersonation, Unknown Key Share, Matching Sessions

1 Introduction

Authenticated Key Exchange (AKE) protocols form a crucial component in many network protocols. In 1993, Bellare and Rogaway [4] presented the first formal security model and security definition for key agreement. The model and associated definitions evolved over the years, as such, they have been subject to increasing requirements in terms of efficiency as well as security. After dozens of years research, some important issues remain without satisfactory treatment. One of the goals has been to design protocols that are secure in the presence of the strongest possible adversaries.

When confronted with different models it is important to select the right model for strongest results that is in itself a non-trivial task. The Canetti and Krawczyk model [11], henceforth called CK01, and the model by LaMacchia, Lauter and Mityagin [23], referred to as eCK, are considered among the most advanced models. The eCK and CK01 security model both adopt the indistinguishability approach of [19] to define security: generally speaking, a key-exchange protocol is called secure if under specified adversarial abilities and security definitions, the attacker is infeasible to distinguish the real value of a session key generated by the protocol from an independent random value. Comparing to static corruption AKE models (such as [4][2]), those two models address leakage of session specific ephemeral data in relation to protocol security. Whereas the respective treatments of leakage are one of the major differences between the two models. With respect to the CK01-adversary who is given a *SessionStateReveal* query to learn session specific private information which is defined in the protocol specifications. On the other hand the eCK adversary

is equipped with $EphemeralKeyReveal$ query to access all ephemeral private input required to carry the session computations. The $SessionStateReveal$ query cannot be issued against the test session; by contrast the $EphemeralKeyReveal$ query can be used against the test session under certain conditions, suggesting that eCK captures a wider range of security attributes than CK01, without leaving out any CK01 implied properties. However, given the subtle differences among the models, this conclusion is not obvious. In fact, many technical differences suggest that the models are formally incomparable, besides the adversarial capability, another important issue is the different application area (e.g., the symmetric-role protocol is not applicable in eCK model comparing to CK01 model). In other word, the relation between strong AKE security models has not been made precise combined with the unproven assumption that some models are stronger than others in practice, hinders the objective comparison of the security properties of the various protocol proposals. There is clearly still a need for a good security definition, since for each model, there exist protocols that can be shown correct in it, but that cannot be shown correct in the other models as discussing in section 2.3.

The core property of the two strong AKE security models is the indistinguishability, which highly depends on the capabilities of the adversary. However it raises other questions: how a security model includes all attacks and what kind of adversary power is reasonable, along with the technology or application evolution. So that, the eCK and CK01 model left over a lot of ambiguities on adversary capabilities in security definitions. In fact, a lot of vulnerabilities are due to the implementation other than the protocol design. Therefore, it is not a good idea to establish 'indistinguishablility' on ruling out some reasonable adversary threats or simply increasing the power of adversary in security definitions. On the other hand, neither eCK model nor CK01 model have clear definitions on the entity authentication (other than BR93 model[4]), which is most important point to design and analysis the AKE protocols. With respect to the CK01 model, which assume that the matching sessions share the same unique session identifier, thus in practical it is required a secure authentication protocol executed before the model can be applied. Whereas the definitions on secure AKE protocols in eCK model are slight variants of CK01 model's, which also do not refer to authentication related issues. That makes it hard to analyze the correctness of an AKE protocol under different adversary capabilities. Please note that a protocol capture indistinguishablility property in certain adversary model does not imply its correctness on entity authentication.

In this paper, we are motivated to formalize the issues on the relationship of authentication timing, key generation and confirmation, to develop a more general model for the analysis of security protocols based on a flexible definition of matching sessions. In particular, we focus on the situations involving adversaries with strong adaptive corruption capabilities (e.g., compromising the states of sessions or long-term private key of a party), and every secret related to session key is possible to be revealed somehow by adversary. Meanwhile, we show that it is especially required for an AKE protocol to provide correct timing of authentication and key generation (Not based on the assumption that some attack is not feasible), to guarantee the integrity and authenticity of communications over adversarially controlled links. The underlying intuition is that provided newer security models are stronger, and hence protocols proven in the newer models are at least as secure as the protocols proven in earlier models. We argue that the stronger models should at least have the same application domain as the previous models, e.g. the symmetrical role protocols should be executable. As another motivation of this paper, we bridge this surprising gap in existing strong AKE security models (i.e., the CK01 and eCK model), by appropriately unifying their adversary models into one framework. We are also with a desire to clarify the ambiguities of those existing security definitions and strengthen powers available to the adversary to obtain new security definitions.

1.1 Related work

In the seminal papers [4] on authenticated key agreement, Bellare et. al. proposed the first complexity-theoretic based model which formalize the security of KE protocols in the realistic setting of concurrent sessions running in an adversary controlled network. Since then, many indistinguishability-based proof models for AKE security have been proposed, e.g., [5][6][2][3][8][32]. Some earlier models have been compared by Choo and Boyd et al, in [13].

With respect to the strong AKE security models such as CK01[11], $\mathrm{CK}_{\mathrm{HMQV}}$ [22], eCK [23], duo to their differences on adversarial capability and application domains make them hard to compare or to relate to practice. There are vastly different views on the subject: [27][33] claim that $EphemeralKeyReveal$ is no weaker than $SessionStateReveal$, whereas [14] argues that $SessionStateReveal$ is stronger. Further comparison for these two queries has been addressed by Ustaoglu in [34], who also pinpoints that that recent analysis utilizing $SessionStateReveal$ and $EphemeralKeyReveal$ have a similar approach to ephemeral data leakage.

Meanwhile, Menezes et al.,[26] introduce a new query referred to $EphemeralPublicKeyReveal$, to model the leakage of ephemeral public key before session run. They also enhance the eCK security models and definitions to encompass a class of protocols that are executable and secure in both the pre- and post-specified peer models.

By contrast with comparing single query, in [15] Cremers et al., have shown that the CK01, eCK, and $\mathrm{CK}_{\mathrm{HMQV}}$ models for AKE security are not only formally but also practically incomparable, thereby refuting several (informal) claims made in the literature in e.g. [27][33][23] about the relation between the models.

In 2009, David Basin and Cas Cremers [1], formalize a hierarchy of adversary models for security protocol analysis, ranging from a Dolev-Yao style adversary to more powerful adversaries who can reveal different parts of principals' states during protocol execution. They define our hierarchy by a modular operational semantics describing adversarial capabilities. [30]

In 2010, Sarr et al. [30] proposed a strengthened eCK model which is closed to our work to address the leakage of the intermediate results that a party may need to compute during key exchange procedure or before computing the final session key. They particularly discussed the damage consequences of ephemeral exponents' exposure that might help the adversary impersonate as the victim against other party. However, theirs model is (particularly) suited for distributed implementations wherein a tamper-proof device is involved. It also doesn't formalize the problem about the 'timing' to authenticate and accept a matching session in theirs model, like its predecessors, i.e. eCK and CK01 models.

Other recent similar works, such as [35], in which paper a new mechanism (referrd to non-malleable joint proof-of-knowledge) is proposed for designing specific class of Diffie-Hellman protocols (HMQV-like) in concurrent settings, and discuss the security based on CK model. Whereas our works try to provide a new generic model to formalize the authentication problems, and we seek better way to define and prove the security for different types AKE protocols in particular involving strong adversary capabilities.

1.2 Contribution.

We discuss the ambiguity and inadequacy of existing strong AKE security definitions, mainly concerning the CK01 and eCK model. Our results show that it is important to define or discuss the adversary capabilities on the basis of the correct authentication model. We give a more reasonable AKE model for designing a protocol and analyzing its security while involving strong adaptive corruption adversary capabilities, in which we introduce a flexible definition of matching sessions

(deviate from BR93 model) based on a new type session identifier. In contrast, we formalize the authentication related issues from the beginning of communication, and in particular pay close attention to the timing (and relationship) of the authentication, session key generation and key confirmation processes for different classes of AKE protocols, that would be useful to examine the correctness of an AKE protocol.

In addition we propose a new protocol named Σ^y as an instance to verify our idea, which is proven under our security definitions based on random oracle model and GDH assumption. An interesting point of this protocol is that it is able to protect against KCI [8], UKS [9] and ECI (as Definition 9) attacks by applying a unique identifier of matching sessions in underlying key generation procedure instead of relying on hashing of key material.

Organization. The remaining paper is structured as follows. First, in section 2 show two attacks on two protocols under eCK mode that utilize the leakage of session states, and we discuss the ambiguity and shortcoming of CK01 model and eCK model. In section 3, we define a formal AKE model that captures strong adaptive corruption adversarial capabilities. We introduce the new session identifier in section and definition of matching sessions. The proposed protocol Σ^y and analysis under the proposed security model are given in section 4, and we conclude in section 5.

2 Protocols and Attacks

2.1 NAXOS protocol

The NAXOS protocol, as defined in [23] which satisfies the eCK security definition. The protocol is designed to be secure in a very strong sense: the adversary is assumed to either have the capability of learning long-term private keys, or has the capability of learning short term data generated during a protocol session that does not include the private key. The intuition behind the design of the protocol is that by combining the long-term private key with the short term ephemeral key inside the hash function, the adversary would need to have both of these elements to construct an attack.

In order to start a session, the party $\hat{A}$ is activated as initiator to sends $(\hat{B}, \hat{A}, X = g^{H(\tilde{x},a)})$ to $\hat{B}$, and the latter party responds with $(\hat{A}, \hat{B}, Y = g^{H(\tilde{y},b)})$. Both parties have a long-term private key, and initially know the public key of all other participants. Party $\hat{A}$ computes the key materials $\sigma_a = Y^a$, $\sigma_b = B^{H_1(\tilde{x},a)}$ and $\sigma_c = Y^{H_1(\tilde{x},a)}$; then the session key can be computed as $sk_A = H(\sigma_a, \sigma_b, \sigma_c, A, B)$. Party $\hat{B}$ will compute the same way session key as $sk_B = H(X^b, A^{H_1(\tilde{y},b)}, X^{H_1(\tilde{y},b)}, \hat{A}, \hat{B})$. In [14] Cremers et al. suggest that *SessionStateReveal* could leak the shared secrets key materials (σ_b and σ_c) (for HMQV the shared secret is σ). They modify the original eCK model to eCK' by substituting *EphemeralKeyReveal* with *SessionStateReveal*, and then devise two NAXOS attacks on initiator and responder (resp.) with conclusion that *SessionStateReveal* is the stronger query. We construct an unknown key share attack involving three parties in the same sense to show that NAXOS is not secure in CK01 model. The attack proceeds as follows:

1. The adversary register an arbitrary identity $\hat{E}$ with public key $E = Ag^{-c}$, in which A is the public of an honest party $\hat{A}$ and constant c is selected by adversary which ensure E has not been registered in CA.[1]
2. $\hat{A}$ starts an initiator instance, wanting to communicate with $\hat{B}$.

[1] the eCK model allow arbitrary key registration without prove the knowledge of relevant private key, so that we simply assume all public key is unique in CA.

3. $\hat{A}$ chooses her ephemeral private key $\tilde{x}$, and sends out $(\hat{A}, \hat{B}, X = g^{H_1(\tilde{x},a)})$ to $\hat{E}$. The adversary selects this session as test session.
4. The adversary intercept the message $(\hat{A}, \hat{B}, X)$, and then substitute the sender field with identity $\hat{E}$ and sends $(\hat{A}, \hat{B}, X)$ to party $\hat{B}$.
5. $\hat{B}$ starts an responder instance, chooses her ephemeral key $\tilde{y}$, and sends out $(\hat{B}, \hat{E}, X, Y = g^{H_1(\tilde{y},b)})$. The adversary learns this message.
6. $\hat{B}$ computes the session key $k_b = H_2(E^{H_1(\tilde{y},b)}, X^b, X^{H_1(\tilde{y},b)}, \hat{E}, \hat{B})$.
7. The adversary sends the message $(\hat{B}, \hat{A}, X, Y)$ to $\hat{A}$ on behalf of $\hat{B}$.
8. $\hat{A}$ computes the session key $k_a = H_2(Y^a, B^{H_1(\tilde{x},a)}, Y^{H_1(\tilde{x},a)}, \hat{A}, \hat{B})$. During the computation of k_b, the adversary learns the key materials $(E^{H_1(\tilde{y},b)}, X^b, X^{H_1(\tilde{y},b)})$ via $SessionStateReveal$. The adversary first computes $A^{H_1(\tilde{y},b)} = E^{H_1(\tilde{y},b)}Y^c$ and then combines $(A^{H_1(\tilde{y},b)}, X^b, X^{H_1(\tilde{y},b)})$ with the names $\hat{A}$ and $\hat{B}$, and applies H_2, resulting in k_a.

Eventually, The party $\hat{B}$ correctly believed in talking to party $\hat{E}$, but $\hat{A}$ mistakenly believed in communicating with honest party $\hat{B}$. The above sequence forms an unknown key share attack on NAXOS protocol, because the adversary $\mathcal{A}$ can make non-matching sessions (i.e., two initiator session) generate the same key materials $(\sigma_a, \sigma_b, \sigma_c)$ and $\mathcal{A}$ can learn the key of the test session by revealing the local state of the session held by honest party $\hat{B}$ which is not partner of test session. Furthermore, the test session is fresh according to freshness definition of eCK' model[14] because: (1) neither $\hat{A}$ nor $\hat{B}$ are adversary-controlled, (2) no $SessionKeyReveal$ queries are performed on test session (3) no matching session exists, and (4) no long-term private keys. Therefore, the unknown-key share attack has been successfully mounted.

2.2 Protocol $\mathcal{P}$

We review a two-pass Diffie-Hellman key agreement protocol $\mathcal{P}$ presented in earlier version of [15] and [30], as instance. The protocol $\mathcal{P}$ is a signature based NAXOS transform protocol that can be proven secure in the eCK model if the signature scheme is secure against chosen message attacks. The protocol is informally depicted as following:

1. The initiator party $\hat{A}$ performs the steps:
   (a) Choose an ephemeral private key $\tilde{x} \in_R Z_q$, then compute $x = H_1(\tilde{x}, a)$ and $X = g^x$.
   (b) Compute signature $\delta_{\hat{A}} = SIG.Sign(a, \hat{B}||\hat{A}||X)$.
   (c) Send $(\hat{B}, \hat{A}, X, \delta_{\hat{A}})$ to $\hat{B}$.
2. Upon receiving $(\hat{B}, \hat{A}, X, \delta_{\hat{A}})$, the responder party $\hat{B}$ does the following:
   (a) Verify that $X \in Z_q$.
   (b) If $SIG.Verify(A, \delta_{\hat{A}}, \hat{B}||\hat{A}||X) = 0$ then abort with failure.
   (c) Choose an ephemeral private key $\tilde{y} \in_R Z_q$.
   (d) Compute $y = H_1(\tilde{y}, b)$ and $Y = g^y$.
   (e) Compute signature $\delta_{\hat{B}} = SIG.Sign(b, \hat{A}||\hat{B}||X||\delta_{\hat{A}}||Y)$.
   (f) Send $(\hat{A}, \hat{B}, X, Y, \delta_{\hat{B}})$ to $\hat{A}$.
   (g) Compute $k = H(X^y)$.
3. Upon receiving $(\hat{A}, \hat{B}, X, Y, \delta_{\hat{B}})$, the responder party $\hat{B}$ does the following:
   (a) Verify that $Y \in Z_q$.
   (b) If $SIG.Verify(B, \delta_{\hat{B}}, \hat{A}||\hat{B}||X||\delta_{\hat{A}}||Y) = 0$ then abort with failure.
   (c) Compute $k = H(Y^x)$.

Unfortunately, the protocol $\mathcal{P}$ is insecure if the ephemeral DH exponents is somehow exposed to adversary. Consider the attack provide that the adversary $\mathcal{A}$ learned the exponent x of initiator $\hat{A}$'s the ephemeral public key X and intercepted corresponding signature $\delta_{\hat{A}}$, then $\mathcal{A}$ is able to always replay the first message $(\hat{B}, \hat{A}, X, \delta_{\hat{A}})$ to impersonate as $\hat{A}$ to $\hat{B}$ (no matter what is responded from $\hat{B}$). At the same time, note that the session establish at $\hat{B}$, e.g. $s_{\hat{B}} := (\hat{B}, \hat{A}, X, W)$ is unexposed in terms of eCK's freshness definition, although no matching session exists. Then the adversary can select $s_{\hat{B}}$ as test session to distinguish the session key.

Similar attack can also be extended to HMQV protocol[22]. As for a session $(\hat{A}, \hat{B}, X, Y)$, the key material at party $\hat{A}$ is computed as $\sigma = (YB^E)^{x+aD}$, where $E = H_e(Y, \hat{A})$ and $D = H_e(X, \hat{B})$. If the exponent $x + aD$ is compromised during key computation, then the adversary could replay the message to launch impersonation attack. In practice, it is important to confound the damage of ephemeral key leakage even when this leakage involving the exponents of DH key. Since in any environment, it is not guaranteed that leakages on DH exponents cannot happen.

2.3 Discussion

The attacks described above and besides those in[14] [30] have showed that the leakage of local state (say the inputs to the final hash function are part of the local state) may induce more harmful consequences than exposure of long-term private key in AKE protocol and easier to put in practice (e.g., in presence of malware) . At first glance, that eCK model is stronger than CK01 model, because it allows to reveal the ephemeral private key of test session. However, those protocols proven secure in eCK model are not secure in CK01 model. And as [34] pointing out, if $H(\tilde{x}, a)$ is kind of ephemeral key, the NAXOS-like protocol are no longer secure.

Ambiguity of Ephemeral Session States. In one hand, a drawback of the *SessionStateReveal* query in current security models is that the query is often underspecified. For example, in the CK01 model, *SessionStateReveal* is defined as giving the adversary the internal state of the Turing machine that executes the protocol. This internal state is not defined within the security model. Effectively, the definition of the internal state is postponed to the proof of a particular protocol. If the $H(\tilde{x}, a)$ used by the protocol $\mathcal{P}$ is not defined within the *SessionStateReveal*, then it also can be proven secure in CK01 model but with incorrect authentication forms of BR93 model. On the other hand, In eCK model, LaMacchia et al. claim that the *Long-termKeyReveal*, *SessionStateReveal* and $SessionKeyReveal$ queries together can be functional equivalent to *Corrupt* query in CK01 and the latter is no longer necessary; they also replace *SessionStateReveal* with $EphemeralKeyReveal$ query and require that an ephemeral secret key of an AKE session should contain all session-specific information used by a party in the AKE session. We need to see that if $EphemeralKeyReveal$ query return the full internal state of the Turing machine executing the protocol that appears to imply that the adversary can learn the shared secrets computed in the test session in case the response trivially reveals the session key. However, if $EphemeralKeyReveal$ is not allowed against the test session, then the resulting model is weaker in the sense that the adversary cannot obtain any test session specific secrets. Furthermore, if the $EphemeralKeyReveal$ query applying to a session reveals only the ephemeral key $\tilde{x}$ as the proof of NAXOS implied under eCK model from[23], then there is no query models the leakage of local states during each cryptographic operations. That is one of the reasons that the eCK model is incomparable to CK01 model, if the CK01 adversary is able to reveal the full internal states of sessions (except the test session and its matching session) via *SessionStateReveal*. In [30], they classify the ephemeral secrets according to two kinds of implementation schemes (particularly) involving some tamper-proof device (e.g. smart

card). However, the storage medium in the tamper-proof device (e.g memory) for those intermediate results might not be secure either. It increases more restriction on security assumptions and is somewhat not practical for implementation (i.e., How to guarantee that those key materials cannot be leaked).

Inadequacy Authentication Definition. We need notice the major problem of those attacks working on NAXOS is due to the adversary can make non-matching sessions to generate the same key materials (i.e., the $\sigma_a, \sigma_b, \sigma_e$). In other word, the party accepts the session without ensuring whether or not there exists matching session, under CK01 security experiment. Since the authentication in the eCK model is based on the assumption that parties who complete matching sessions compute the same session key, and it is based on the restriction that those session key related materials cannot be revealed. That rules out the threat that those ephemeral secrets leakage, e.g. the leakage the exponent related to session key which might lead to the impersonation attacks. Whereas CK01 model, the session identifiers *sid* for matching sessions are assumed to be externally provided to the security model, e.g. the application layer. But the *sid* is difficult to be assigned before the protocol run in practice. Thus neither CK01 model nor eCK model clearly define the entity authentication related issue, in particular for the timing of authentication and acceptance of matching sessions. For instance, although the protocol $\mathcal{P}$ is secure in eCK model, it is cannot be shown correct in BR93 model. Because the adversary can always replay the first message $(\hat{A}, \hat{B}, X, \delta_{\hat{A}})$ from the initiator $\hat{A}$, to lead the sessions $(\hat{B}, \hat{A}, X, \delta_{\hat{A}}, *)$ at $\hat{B}$ to accept with no matching session. Please note that a basic principle guiding any security formulation for key-exchange protocols is that the leakage of ephemeral information associated with a particular session does not compromise other non-matching sessions. Hence, the question is that if those key materials of one session have been exposed, then how to make sure that it wouldn't affect the security of other incomplete sessions? We should solve this problem, first based on the correct authentication model, instead of excluding some reasonable adversary capabilities.

3 Strong Adaptive Corruption AKE Security

3.1 Session and its Execution

Session Identifying. Generally speaking, a party can identify a session s via tuple $(ID, role, Comm._{ID}, I\tilde{D}^*, role^*, Comm._{ID^*})$. In which The ID, ID^* is the identifier of the participants executing s, and their corresponding roles are either $initiator(I)$ or $responder(R)$. The order of ID in session identifier is pre-determined according to specific protocol following rules: (i) if the session is executed for asymmetric role application (i.e., $role \neq role^*$, then ID is the initiator and ID^* is her intended peer whose role $role^*$ should be $responder$; (ii) if the session is executed for symmetric role application (i.e., $role = role^*$), then the order of ID is determined by the rule of specific protocol, e.g., according to lexicon order. The $Comm._{ID}, Comm._{ID^*}$ are the list of messages that were sent by party ID and ID^* in execution phase, respectively. Meanwhile, we allow the sessions to generate a unique value as session identifier, assuming stored by variable *sid*, which can be useful for practical application, besides identifying a session after session accepted, etc. Although we reserve the interface in session identifier tuple, in the subsequently work we mainly focus on asymmetric role protocols if no specific instructions.

Session Create. A party $\hat{A}$ can be created via an incoming message to create or update a session. The incoming message has one of the following forms for creation: (i) $(\hat{A}, I, \tilde{B}, R)$; (ii) $(IN, \hat{A}, I, \tilde{B}, R)$, where IN is the message sent to $\hat{B}$ from $\hat{A}$.

Session Update. A party $\hat{A}$ can be activated to update a session via an incoming message of the form $(IN, \psi, \hat{A}, I, Comm._{\hat{A}}, \hat{B}, R, Comm._{\hat{B}})$. Upon receipt of this message, $\hat{A}$ proceed as following:

1. $\hat{A}$ retrieves that she owns an active session identified either by the tuple $(\hat{A}, I, Comm._{\hat{A}}, \hat{B}, R, Comm._{\hat{B}})$. The message is rejected if no such session exists.
2. If such session s exists, firstly $\hat{A}$ updates s by appending message IN to $\hat{B}$'s communication message list if it is valid.
3. If the protocol requires a response by $\hat{A}$, then $\hat{A}$ prepares the required response OUT and the outgoing message $(OUT, \hat{A}, role, Comm._{\hat{A}}, \hat{B}, role^*, Comm._{\hat{B}})$. After sending this message, $\hat{A}$ append OUT to her communication list $Comm._{\hat{A}}$.

Session Accept and Reject. If no further messages will be received and the session communications meet protocol specifications, then the party accepts the session and computed session key. We note that a session can be accepted at one party but not necessarily at its partner. However, before finally accepting a session, it would go through a series of internal execution status transition in terms of protocol specifications, which capture those events (but not limited to) including: validation of authentication, ready to accept the session key ("I believe that only my matching session can compute the same key"). Meanwhile, each session should proceed with some checks on incoming messages, e.g. particular for authentication validation. If any verification of incoming messages fails according to protocol specifications or the session is expired, then the session would be rejected to create or abort with erasing all session specific information (including the session key if it has been computed). We stress that such an abortion occurs before the session identifier can be updated.

Matching Sessions. In order to define correctness and security of an AKE protocol with our proposed session identifier, we give a formal definition of the matching sessions referring to the notations in [26]. If m is a message list then $|m|$ denote the number of its messages recorded. We say that two message list m_1 and m_2 are matched, written $m_1 \cong m_2$, if the first $t = min\{|m_1|, |m_2|\}$ messages of the lists m_1, m_2 are pairwise equal. And we let $m_1 \equiv m_2$ denote the two message list are complete equivalent, namely $|m_1| = |m_2|$ and every messages in lists m_1, m_2 are pairwise equal. Please notice a special case of the *matched* case that two message lists can be said to be matched if one of them is empty, i.e. $|m| = 0$.

Definition 1 (Matching Sessions). *We say a session* $s_j^v := (ID_i, I, Comm._{ID_i}^v, ID_j, R, Comm._{ID_j}^v)$ *is matching to session* $s_i^u := (ID_i, I, Comm._{ID_i}^u, ID_j, R, Comm._{ID_j}^u)$ *if they share the same tuple* (ID_i, I, ID_j, R) *and one of the following conditions is satisfied:*

- s_j^v *updated by receiving a message, and it holds that* $Comm._{ID_i}^v \equiv Comm._{ID_i}^u$ *and* $Comm._{ID_j}^v \equiv Comm._{ID_j}^u$.
- s_j^v *updated by sending a message, and it holds that* $Comm._{ID_j}^v \cong Comm._{ID_j}^u$ *and* $Comm._{ID_i}^v \equiv Comm._{ID_i}^u$.

We say that the two sessions s_i^u *and* s_j^v *are matching, if* s_j^v *is matching to* s_i^u*, and vice versa.*

Semantic Match. The 'Semantic' here about matching sessions, involves the identity of participants with corresponding roles and the timing (while send or receive) of match, which are related to authentication. We stress that considering the matching sessions (conversations) without the *sematic* factors is not correct. With respect to the definition of matching conversations in BR93 model (even its variants), the ID and roles have been included in the notation of oracle, e.g. two

oracles $\Pi^s_{\hat{A},\hat{B}}, \Pi^t_{\hat{B},\hat{A}}$ having matching conversations, where $\hat{A}$ could be initiator and $\hat{B}$ is responder according to the order messages exchanged. Whereas the timing of matching (two sessions have matching conversations) in BR93 model is defined at the point when the last message of the protocol has been sent (or received) and the whole communication transcript of the two sessions are equal. However, for most AKE protocol, a session would accept the event on existence of matching session (nor final accept a session) before sending the last message, otherwise it couldn't send the last message. In particular for the AKE protocols involving the key confirmation messages, which are sent after accepting the existence of matching (e.g. when the authentication is valid) and key generation. In other words, the BR93-like definitions on matching conversations don't model the relationship of authentication, key generation and confirmation. By contrast, in our definition of matching sessions, we model the situation that a session has matching session before sending or receiving the last messages. However, this loose restriction result in another problem on the event 'multiple-matching' (namely there exist multiple sessions are matching to a session at some point), we will discuss it in the section 3.3.

3.2 Adversarial Capabilities

In this sub-section we focus on the adversarial model, in which the parties modeled as probabilistic polynomial time (in some security parameter λ) Turing machines (PPTs) and one special PPT $\mathcal{A}$ called adversary. Each party has an identifier $ID \in \{0,1\}^\lambda$ where λ is a system parameter, and a long-term authentication key (e.g., for symmetric or asymmetric authentication) which is used to prove the identity of the party in a protocol.

The active adversary $\mathcal{A}$ with full control of the communication, can intercept and modify messages sent over these links, Who can also schedule or prevent their delivery, inject its own messages, interleave messages from different sessions, etc. In addition, in order to model potential disclosure of session relevant secrets, the attacker is allowed access to ephemeral secrets contained in session state during executing the protocol. Furthermore, the ephemeral public key should also be accessible to the adversary before the session becomes alive. Each participant $\hat{A}$ is given as an oracle to which an adversary has access, basically providing the interface of the protocol. The adversary is allowed to make the following oracle queries:

1. **Send**$(\hat{A}, s, m)$. $\mathcal{A}$ sends any message m to session sid established by party $\hat{A}$ on behalf of other party. The oracle will respond according to the protocol description.
2. **EstablishParty**$(\hat{A}, A)$. This query allows $\mathcal{A}$ to register an identifier $\hat{A}$ and a static public key A on behalf of a party. The adversary totally controls that party, thus permitting the modeling of attacks by malicious insiders. Parties that were established by $\mathcal{A}$ using $EstablishParty$ are called corrupted or adversary controlled.
3. **SessionStateReveal**$(\hat{A}, s)$. The attacker provides the name of a party and a session identifier of a yet incomplete session at that party and receives the internal state of that session. The most important point here is what kind of information included in session states, and it exactly depends on design and implementation of specific KE protocol. Without loss of generality, we classify the session related states as public state (PS) of session and local state(LS) of subroutines. Generally, the PS contains: global variable which could be used for different subroutines (e.g., the outside input of session s), vulnerable randomness generator[2] and the pre-computed ephemeral public key, security parameters of each session, inputs and outputs of subroutines, etc. Whereas the LS is protocol specified and defined within each subroutine in terms of the

[2] We here assume that the randomness generator implemented on host machine is vulnerable, in particular in presence of malware.

security level. Typically, the revealed information will include the public state (PS) of session and all the local state of its subroutines, except for the subroutines labeled with high security level (e.g., signature, symmetric encryption scheme and the key generation subroutine). Nevertheless, we should treat the exposure of PS and LS in different way, because some secrets leakage modeled by other queries may be caused by the exposure of internal local state of some subroutine. In particular, the exposure of intermediate results in the key generation subroutine is equivalent to session key leakage. Hence we attribute those event to corresponding queries (e.g., $SessionKeyReveal$ or $Long\text{-}termKeyReveal$) other than $SessionStateReveal$. Therefore, we mainly focus on the session states related to protocol design in this query, namely the public state of session mentioned above. Accordingly, we also assume the storage mediums for public states are less secure than the local states'. Another important issue is the timing of states leakage and if the session would proceed after exposure. Basically the adversary is allowed to issue the $SessionStateReveal$ query at any point with many times. And in normal situation the session will continue to process after public states exposed, except for the adversary reveal those states in specific way (e.g., memory freezing attack).

4. ***Long-termKeyReveal***$(\hat{A})$. This query returns the private key of party $\hat{A}$ without full control of the party. For instance, signature scheme like ElGamal [18],Schnorr [31], etc., would expose long-term private key if the random bits used for computing signature was leaked.
5. **SessionKeyReveal**$(\hat{A}, s, aux)$. Given a party's name, a session identifier of a session and an auxiliary input aux, this query returns the key generated by the named session or the internal local states of key generation subroutine that have not been erased (incl., the key materials[3] including prepared exponent, etc). Without loss of generality, we specify the adversary to issue this query only once since the session has been turned into key generation process (typically, key generation subroutine might be invoked while receiving intended messages, e.g. valid authentication message). Please note that all secrets used during the key generation are ephemeral and should be erased immediately after using, hence there is no further output would be obtained from this query after the point of this query being issued. Whereas the parameter aux is used to capture the timing of issuing this query in terms of the secrets interesting the adversary, and the value of aux should be specified by each KE protocol. This attack provides the formal modeling for leakage of session keys related information that may result from events such as break-ins, cryptanalysis, careless disposal of keys, etc. We also model the side-channel attack as part of $SessionKeyReveal$. For example while considering a protocol involving the Diffie-Hellman key, the leakage of exponent $s_A = x\psi + a$ during key computing might directly result in session key exposure; note that those DH exponents are also part of ephemeral internal local states of key generation subroutine. We also assume the DH exponent operation using the same algorithm, hence revealing the DH exponent x while computing g^x is equivalent to revealing the s_A on technique even if there is involved some tamper-proof device, e.g. via the power consumption analysis [21]. And we stress the storage medium of those DH exponents should be treated as the same security level as session key while implementation.
6. **Corrupt**$(\hat{A})$. The attacker can decide at any point to corrupt a party, in which case the attacker learn the whole current internal memory of that party including all session-specific public states and the full internal local states of all subroutines; reveal corresponding long-term secrets (such as private keys or master shared keys used across different sessions) or even replace the party's long-term secrets with her own choice. Since then the corrupted party is considered completely controlled by the attacker from the time of corruption and the attacker can impersonate the

[3] Such as the $\sigma_a, \sigma_b, \sigma_e$ in NAXOS protocol

party in all its actions departing arbitrarily from the protocol specifications. If a party is not corrupted it is said to be honest.

Definition 2. *A probabilistic polynomial-time adversary with the above capabilities is called a KE-adversary.*

Remark 1. We say that the adversary $\mathcal{A}$ adaptively corrupts sessions, if $\mathcal{A}$ issued *SessionState-Reaveal*, and *SessionKeyReveal* queries on her own choice. The session's corruption doesn't imply that the parties have been corrupted. However, a party can be corrupted via either *EstablishParty* or *Corrupt*, in different way. Although those two queries have some common feature (i.e., choose the long-term key on behalf of a party), the major difference is that the *Corrupt* query can be only issued on established party (namely the party must be exist before); whereas the *EstablishParty* query focus on registering a new party according to attacker's choice. We stress that the exposed sessions or the leakage of long-term private key at some point shouldn't affect the incomplete fresh sessions.

3.3 Security of AKE Protocols

AKE Protocols. Normally, there have three kinds of AKE protocols when consider the relation between the key-generation and authentication: (i) authentication and then key-generation (AatK), e.g. the SSL/TLS protocol [17][16]; (ii) key-generation and then authentication (KatA), e.g. the 4-pass Σ_0 protocol [12] and the generic compilers for AKE in [20]; (iii) key-generation integrated with authentication (KIA), e.g. the (H)MQV protocol [24][22] or the protocol in [10].

Timing of Authenticatoin and Key generation. We discuss the authentication issue of AKE protocol based on our Definition 1 of matching sessions (following the methodology of BR93 model[4]). Let **Multiple-Match**$^{\mathcal{A}}(\lambda)$ denote the event that there exist at least two distinct sessions s_j^v and s_j^w, which are matching to s_i^u. Note that the notation **Multiple-Match**$^{\mathcal{A}}(\lambda)$ here is different from BR93's, because our definition of matching sessions allow at some point there are multiple sessions are matching to a session. We further assume that each session keep variables $\Lambda \in \{$*pre-accept-key*, *accept-matching*, *accept-session*, *reject-session*$\}$ to track the execution status (as public states) of the protocol, which is initialized to be string "null". Where the *pre-accept-key* denotes the event that a session has computed the session key and is ready to accept the key without proceeding with the final formal key confirmation. The *accept-matching* denotes the event that a session accepts the existence of matching session. The *reject-session* represents the event that a session is rejected at some point. And the *accept-session* denotes the event that a session and corresponding session key are accepted. We stress that while a session being in *accept-session* it must has gone through the execution stage of *pre-accept-key* and *accept-matching*. Typically, the execution status of a session is set be *pre-accept-key* if either the authentication procedure is valid or the key is generated by a $KatA$ protocol. Let KG, KGA, Aut, Con denote respectively the processes: *key generation* in which the session key is computed; *the key generation integrated with implicit authentication*; *explicit authentication*, e.g. signature or symmetric encryption scheme etc.; and *key confirmation*, e.g. the message authentication code scheme. We informally present the execution status transitions for each class of protocols as following.

- AatK: $\xrightarrow{Aut}$ *accept-matching* $\xrightarrow{KG}$ *pre-accept-key* $\xrightarrow{KC}$ *accept-session*.
- KatA: $\xrightarrow{KG}$ *null* $\xrightarrow{Aut}$ *accept-matching*, *pre-accept-key* $\xrightarrow{KC}$ *accept-session*.
- KIA: $\xrightarrow{KGA}$ *pre-accept-key* $\xrightarrow{KC}$ *accept-matching*, *accept-session*.

Proposition 1. *For a two party AKE protocol with the session identifier comprising the concatenation of exchanged messages according to the ID of participants and their corresponding roles, if a party execute a session without pre-shared states and with existing only benign adversary, and at some point its communication transcript contains the unpredictable random values (in terms of protocol specifications) selected by itself and its matching session, then the session has a unique matching session except with negligible probability, i.e. the probability of* $\textbf{\textit{Multiple-Match}}^{\mathcal{A}}(\lambda)$ *is negligible.*

Proof. Since the benign adversary only honestly relays the communications among sessions, when two honest parties execute matching sessions then theirs exchanged messages are semantic matching in terms of ID and roles. Assuming two sessions s_i^u and s_j^v are matching and the transcript of communications contains the values selected uniformly at random on a per-session basis (e.g. the ephemeral public key X, Y), then each session has a unique matching session except with negligible probability.

While considering the security in real word, a more complex model in required to model the capabilities of active adversaries (i.e. the *KE-adversary* as Definition 2) to define secure authenticated key exchange. In which situations the explicit or implicit authentication schemes, say, symmetric encryption (e.g., IND-CPA or IND-CCA security), or digital signature schemes (e.g., EUF-CMA), might be involved, thus we hereby give a formalism of authentication timing for different class of AKE protocols.

Definition 3 (Unique-pairwise Matching Sessions). *We say two sessions* s_i^u *and* s_j^v *executed by honest party, are unique pairwise matching, if and only if* s_i^u *is the unique matching session to* s_j^v *and vice versa. Namely the probability of* $\textbf{\textit{Multiple-Match}}^{\mathcal{A}}(\lambda)$ *on both* s_i^u *and* s_j^v *are negligible.*

Definition 4 (Pre-SessionKey). *The pre-sessionkey is a set of key materials used to directly compute the session key in a Cryptographic operation, which is generated within the session key generation subroutine and labeled secret. More specifically, the pre-sessionkeysatisfy the following requirement: (i) distributed uniformly at random, (ii) computed by no more than unique-pairwise matching sessions and distinct among non-matching sessions.*

Proposition 2. *The correct authentication timing for an AKE protocol should be at or after the point when a session has a unique matching session. And two sessions without pre-shared state, can't accomplish authentication in two pass.*

So that we informally have the implications: (i) *pre-accept-key* $\nRightarrow$ *accept-matching*, (ii) *accept-matching*$\Rightarrow$*pre-accept-key*. Since the *pre-accept-key* stands for that a session believes only its unique matching session can compute the same session key. However, if the session key is generate by a KIA protocol instance that the authentication relies on the computed key, then without further key confirmation process the session cannot reach *accept-matching*.

We now define the security for AKE protocols by running an experiment between a *KE-adversary* $\mathcal{A}$ and a challenger $\mathcal{C}$ with a set of parties, in which each party can run a (polynomially-bounded) number of sessions implementing the considered AKE protocol. The goal of adversary is to differentiate the real session key k of a fresh session from a random value, to break the indistinguishability property.

Definition 5 (Freshness). *Let* s *be a completed session held by a party* $\hat{A}$ *with other party* $\hat{B}$*, and both parties* $\hat{A}$ *and* $\hat{B}$ *are not corrupted. Meanwhile,* $\hat{B}$ *supposedly engaged in a session* s^**, such that* s *and* s^* *(if it exists) are unique-pairwise matching. Then the session* s *is said to be fresh if none of the following conditions hold:*

1. *The adversary $\mathcal{A}$ issued either $SessionKeyReveal(\hat{A}, s, aux)$, or $SessionKeyReveal(\hat{B}, s^*, aux)$, if s^* exists.*
2. *If session s^* exists, $\mathcal{A}$ issued one of following:*
   (a) *both Long-termKeyReveal($\hat{A}$) and SessionStateReveal(s).*
   (b) *both Long-termKeyReveal($\hat{B}$) and SessionStateReveal(s^*).*
3. *If session s^* not exists, $\mathcal{A}$ issued one of following:*
   (a) *both Long-termKeyReveal($\hat{A}$) and SessionStateReveal(s).*
   (b) *Long-termKeyReveal($\hat{B}$).*

Definition 6 (Indistinguishability Security Experiment). *In the experiment, the following steps are performed:*

1. *The KE-adversary $\mathcal{A}$ may issue polynomial number of queries as aforementioned in the security parameter, namely $\mathcal{A}$ makes queries: Send, EstablishParty, SessionStateReveal, Long-termKeyReveal, SessionKeyReveal and Corrupt.*
2. *At some point, the adversary issues a **Test**(s) query on a fresh session s during the experiment with only once. If the session key of s has been computed, then this query is initialized with a random bit b, and it returns the real session key if $b = 1$, or a session key drawn uniformly random from the session key distribution if $b = 0$; otherwise, it returns with failure. $\mathcal{A}$ can continue interacting with the sessions at will after issuing the Test query until their internal execution status Λ = pre-accept-key, but must ensure that the test session remains fresh throughout the experiment.*
3. *At the end of the experiment, the $\mathcal{A}$ terminates with outputting a bit b' as its guess for b where b is the random bit chosen in the Test query.*

We say an adversary $\mathcal{A}$ wins the experiment if the following conditions hold:

- *$\mathcal{A}$ has issued a Test query on a session s^t.*
- *The session s^t has internal execution status Λ = pre-accept-key.*
- *The guess bit b' of $\mathcal{A}$ is equal to the bit b from the Test query (i.e. $b = b'$).*

Remark 2. We stress that the indistinguishability security experiment don't include the communications in the period from the status *pre-accept-key* to *accept-session*. Since the goal of AKE protocol is to establish secret keying information for honest parties, and then build up confidential communication channel. While a session being in *pre-accept-key* that means the parties has already believed that the generated key can be only computed by unique-pairwise matching sessions. Therefore a protocol should ensure that the key is secure (indistinguishable) before final confirming it. Moreover, the final confirmation might carry out confirmation implicitly by using the key to encrypt later communications which might contain some known plaintext.

Definition 7. *We say an AKE protocol Π is secure, if for all KE-adversaries $\mathcal{A}$ running against Π, it holds that:*

1. *If a session s_i^u executed by an uncorrupted party, then except with negligible probability one of the following conditions are satisfied:*
   - *When s_i^u has internal execution status Λ = pre-accept-key, then it outputs a session key which is derived from the pre-sessionkey.*
   - *When s_i^u has internal execution status $\Lambda \in \{$accept-matching, accept-session$\}$, and if it is fresh then there is a session s_j^v such that s_i^u and s_j^v are unique-pairwise matching.*
2. *The probability for the $\mathcal{A}$ in winning the indistinguishability security experiment as Definition 6, is bounded by $\frac{1}{2}$ plus a negligible fraction in the security parameter.*

Remark 3. We stress that the final session key is required to be derived from the *pre-sessionkey*, because the correctness of protocol should provide an assurance that no sessions other than unique-pairwise matching sessions can possibly compute the keying information agreed. This requirement thwart the threat that the adversary establish two non-matching sessions whose session key are computed by the same key materials (in which case the adversary could select one of them as test session and then win the experiment by compromising the other session via *SesssionKeyReveal*). And we recommend that the key for message authentication code scheme should be computed by the *pre-sessionkey* other than the session key. Please note that those key materials of NAXOS or HMQV protocol are not *pre-sessionkey*, although the final session key is unique.

On the other hand, the correctness of protocols must guarantee that having unique-pairwise matching sessions, i.e. $\Lambda =$ *accept-matching*, implies the session key is computed by *pre-sessionkey* and then the status $\Lambda =$ *pre-accept-key*. For instance, if a *AatK* protocol with session identifier as defined in section 3.1, then it should authenticate the session identifier, within the explicit mutual authentication process (e.g., involving the signature scheme). Thus the acceptance of matching sessions ensure that the computed key material, e.g. g^{xy}, is *pre-sessionkey*. And meanwhile the exponents x or y should be generated and stored securely, i.e. not included in public states. Furthermore, in practical application, it is reasonable to suggest that no AKE protocol should be used without key confirmation, although it can be independent (non-related) to the key establishment procedure. An unexposed session s should never be accepted until the session key is carefully confirmed and s has unique matching partner. So that it must involve some key confirmation procedures, say message authentication code, while designing an AKE protocol. Besides, the adopted key confirmation scheme must be secure, that no *KE-adversary* is able to make a fresh session accept without matching session.

In addition, we claim that the protocol is secure in our proposed model implying the security in eCK and CK01 model. The more details about the implication relations are discussed in the Appendix C

4 Protocol Σ^y

In this section we present the Σ^y AKE protocol based on public key infrastructure, informally depicted in Figure 1. The purpose of presenting the Σ^y protocol is to demonstrate that the security definition of §3 is useful (and not too restrictive) in the sense that there exist practical protocols that meet the definition under reasonable assumptions.

Preliminary. In the protocol description, let $\mathbb{G}$ be a multiplicative abelian group and $\mathbb{G}^* = \langle g \rangle \in \{0,1\}^\lambda$ be a subgroup of $\mathbb{G}$ of prime order q, where λ be the security parameter. A party $\hat{A}$ is supposed to register its public key $pk = A$ at certificate authority (CA) and keeps corresponding long-term secret key sk privately (e.g., $A = g^a, sk = a$); after registration, any other party can securely obtain this registered public key with a signed certificate. We assume that parties can register arbitrary keys at CA without proving the knowledge of corresponding private key. Also the adversary can register keys for corrupted parties at any time.

Let $H_1, H_2, H_3 : \{0,1\}^* \to \{0,1\}^\lambda$ are hash functions. The Computational Diffie-Hellman (CDH) assumption in $\mathbb{G}$ is that computing $CDH(U,V) = g^{uv}$ is infeasible given $U = g^u$ and $V = g^v$ where $u, v \in_R Z_q$. The Decisional Diffie-Hellman (DDH) assumption in $\mathbb{G}$ is that distinguishing DH triples (g^a, g^b, g^{ab}) from random triples (g^a, g^b, g^c) is infeasible. The Gap Diffie-Hellman (GDH) assumption [28] in $\mathbb{G}$ is that the CDH assumption holds even when a CDH solver is given a DDH oracle that distinguishes DH triples from random triples. Please note that as the definition of the

matching sessions, when a session updates the session by receiving a messages then the session identifier must be totally equal to its matching session. Then we could compute a session identifier transformation, here denoted as ψ, for identifying the unique-pairwise matching sessions in the sense of Definition 3.

Lemma 1. *For a two party AKE protocol with the session identifier as in Section 3.1, if a honest party update a session s_i^u with receiving a message supposedly from the session s_j^v, and at this point its communication lists $Comm._{ID_i}^u$ and $Comm._{ID_j}^u$ both contain the unpredictable random values (in terms of protocol specifications) selected by each session, then except with negligible probability the session s_i^u has a global unique session identifier which is only shared with its unique matching partner s_j^v if it exists, and distinct to non-matching sessions.*

Proof. First if the communications between s_i^u (assuming it's initiator) and s_j^v are relayed by benign adversary, then according to the definition of matching session, they would share the same identifier $(ID_i, I, Comm._{ID_i}^u, ID_j, R, Comm._{ID_j}^u)$ while s_i^u update its session identifier. When $Comm._{ID_i}^u$ and $Comm._{ID_j}^u$ both contain values selected at random by each session, then in the sense of Proposition 1, s_i^u and s_j^v are unique-pairwise matching, and share the same global unique session identifier. On the second, if the list $Comm._{ID_j}^u$ has messages assigned by the active *KE-adversary*, then the matching session s_j^v is not existent at party ID_j. However, in which case, because of the $Comm._{ID_i}^u$ including the random valued selected by s_i^u on per session basis, its session identifier is still unique among non-matching sessions.

4.1 $\sum^{y}$ description

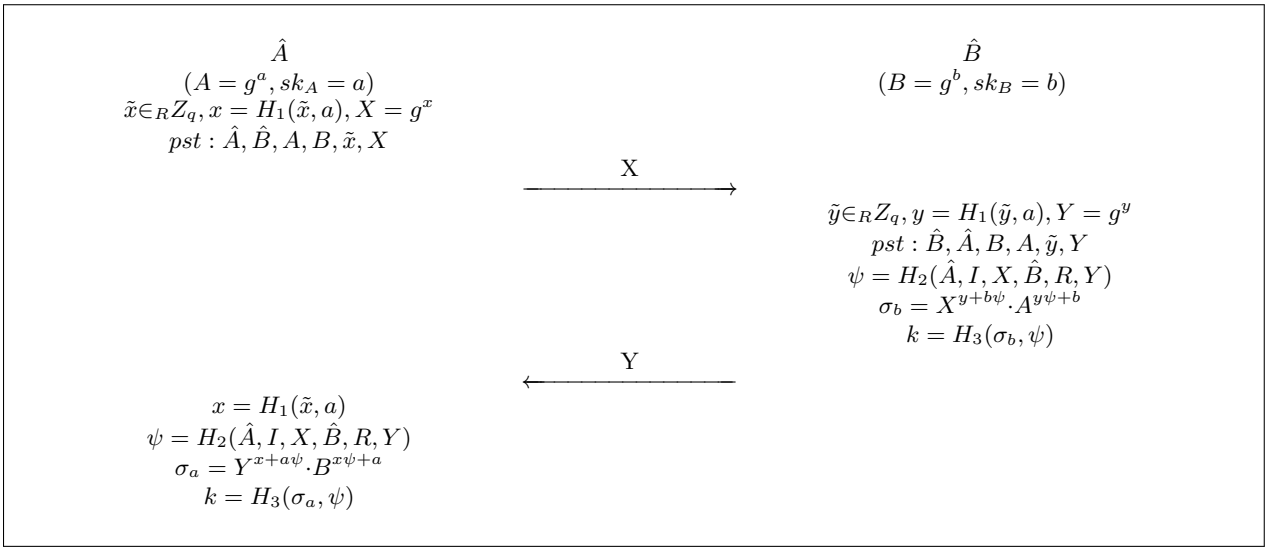


Fig. 1: Protocol Σ^y under the model in section 3.

Definition 8 (Σ^y). *For asymmetric application the protocol proceeds as follows:*

1. *Upon activation a session $(\hat{A}, I, \hat{B}, R)$, the initiator party $\hat{A}$ performs the steps:*
   (a) *Choose an ephemeral private key $\tilde{x} \in_R Z_q$, then compute $x = H_1(\tilde{x}, a), X = g^x$, and erase x.*

(b) *Create an active session with identifier* $s : (\hat{A}, I, X, \hat{B}, R)$.
(c) *Create public session state that contains:* $\{\tilde{x}, \hat{A}, I, X, \hat{B}, R\}$.
(d) *Send* $(X, \hat{A}, I, \hat{B}, R)$ *to party* $\hat{B}$.

2. *Upon receiving* $(X, \hat{A}, I, \hat{B}, R)$*, the responder party* $\hat{B}$ *does the following:*
(a) *Verify that* $X \in Z_q$.
(b) *Choose an ephemeral private key* $\tilde{y} \in_R Z_q$*, and create public session state that contains:* $\{\tilde{y}, \hat{A}, I, X, \hat{B}, R, Y\}$.
(c) *Compute* $y = H_1(\tilde{y}, b), Y = g^y$*, and erase* y.
(d) *Compute* $\psi = H_2(\hat{A}, I, X, \hat{B}, R, Y)$.
(e) *Create an active session with identifier* $s : (\hat{A}, I, X, \hat{B}, R, Y)$.
(f) *Update public session state that contains:* $\{\tilde{y}, \psi, \hat{A}, I, X, \hat{B}, R, Y\}$.
(g) *Compute* $y = H_1(\tilde{y}, b)$.
(h) *Compute pre-sessionkey* $\sigma_b = X^{y+b\psi} \cdot A^{y\psi+b}$.
(i) *Erase* $y, X^{y+b\psi}, A^{y\psi+b}$.
(j) *Compute* $k = H_3(\sigma_b, \psi)$.
(k) *Set execution status to be pre-accept-key and erase* $\sigma_b, \tilde{y}$.
(l) *Send* $(Y, hatA, I, X, \hat{B}, R)$ *to party* $\hat{A}$.

3. *Upon receiving* $(Y, \hat{A}, I, X, \hat{B}, R)$*, the* $\hat{A}$ *does the following:*
(a) *Verify that* $Y \in Z_q$.
(b) *Compute* $\psi = H_2(\hat{A}, I, X, \hat{B}, R, Y)$.
(c) *Update session to* $s : (\hat{A}, I, X, \hat{B}, R, Y)$.
(d) *Update the public session state that contains:* $\{\tilde{x}, \psi, \hat{A}, I, X, \hat{B}, R, Y\}$.
(e) *Compute* $x = H_1(\tilde{x}, a)$.
(f) *Compute pre-sessionkey* $\sigma_a = Y^{x+a\psi} \cdot B^{x\psi+a}$.
(g) *Erase* $x, Y^{x+a\psi}, B^{x\psi+a}$.
(h) *Compute* $k = H_3(\sigma_a, \psi)$.
(i) *Set execution status to be pre-accept-key and erase* $\sigma_a, \tilde{x}$.

If any of the verifications fail, the party erases all session-specific information and marks the session rejected.

Whereas the protocol execution for symmetric role's application is analogous. In contrast, the difference in description is that the party $\hat{A}$ *and* $\hat{B}$ *could only execute the same step 1, 2 (described above) without the step 2l, respectively in symmetric manner. Namely, we need to replace the identifier* $\hat{A}$ *with* $\hat{B}$ *in step 1 while illustrating the behavior of* $\hat{B}$*, and replace the identifier* $\hat{B}$ *with* $\hat{A}$ *in step 1 while illustrating the behavior of* $\hat{A}$*; and set the roles in session id to be the same for both parties.*

4.2 Security Analysis of Σ^y

Remarks on protocol design and implementation. We highlight the following aspects of protocol Σ^y design.

– Session identifier transformation ψ. The ψ is computed in a session s when there at most one matching session to s. We don't specify how the ψ be computed in the session identifier definition. In practice a protocol could include the hashing of last successful generated session key shared. We stress that this kind of session identifier implementation is generic, which could also be applied to other protocols (e.g., MQV) to enhance the security, as presenting in Appendix B. When a session accepted, if there is some requirement to identify the 'IN' and 'OUT' communications separately in established confidential channel, then one can split the ψ into two

parts for identifying the unique-pairwise matching sessions respectively. Moreover, a notable issue on implementation is the length of ψ, as pointed out in [25][26], if the bitlength of ψ is short (say only 80), then the protocol might be vulnerable to online-identity selection attack. Since the ephemeral public key might be computed and leaked before the session is created, the adversary might find hash collision on some identity and register the identity together with victim's public key at CA. Then the adversary could break the security by establishing two non-matching sessions which compute the same session keys.

- The $\tilde{x}$ versus x. We consider the leakage of ephemeral secrets from three casesWe consider the leakage of ephemeral secrets from three cases: generation, computation and storage. While designing the protocol Σ^y, it is reasonable to assume that the randomness generator implemented on host machine is vulnerable, e.g. a flaw in the Debian Linux OpenSSL implementation[7], and only the long-term private key is carefully stored (e.g. tamper-proof device). So that the $\tilde{x}$ might be somehow leaked when it is deployed on host machine without extra secure device. However, we treat the subroutines contained the operation: $H_1(\tilde{x}, a)$,[4]as the same security level as key generation (which will compute the intermediate result:$H_1(x, a)$, either). Hence, the $x = H_1(\tilde{x}, a)$ should not be included in public states and erased immediately after using. In this sense, the derivation of x is an universal way to thwart the threat of ephemeral randomness leakage (i.e. the $\tilde{x}$), because we don't need extra secure module(e.g. smart card) to generate the $\tilde{x}$ while realizing the protocol because it is really costly.
- Key generation. During key generation, we stress that no more than one of the secrets: $x, x + a\psi, x\psi + a, \sigma, k$, can be revealed via $SessionKeyReveal$, unless the session owner ($\hat{A}$) has been corrupted. In other words, we require the party don't keep those secrets simultaneously when computing the session key. The hashing of key materials used in HMQV or NAXOS implies only matching sessions can generate the same session key and resist with the attacks such as UKS or KCI, etc. Our protocol achieves the same effects by embedding the global unique session identifier into the key materials deviation operation to generate so-called *pre-sessionkey*, before submitting the key materials to final hash function. Informally speaking, concerning those attacks mentioned in section 2, we assume the adversary $\mathcal{A}$ obtained the *pre-sessionkey* σ via $SessionKeyReveal$, in order to replace the session identifier used in the σ, $\mathcal{A}$ has to further learn both the long-term private key and ephemeral private key of the session (it violates the definition of fresh session).

Lemma 2. *Under random oracle model, the session identifier transformation ψ in Σ_y protocol is a global unique value, for identifying the unique-pairwise matching sessions if it exists.*

Proof. Since the random oracles produce no collision, we only need to prove the inputs for oracle H_2 are distinct for non-matching sessions. As shown in the description of Σ^y, the identifier ψ is derived from the de facto session identifier tuple (i.e., messages transmitted, the identifier of participants and their corresponding roles). If those ephemeral public key are chosen uniformly random by unique-pairwise matching sessions in the sense of Proposition 1 and Lemma 1, then the inputs of H_2 is global unique with overwhelming probability among non-matching sessions.

Theorem 1. *If H_1, H_2 and H_3 are modeled as random oracles, and $\mathbb{G}$ is a group where the gap DH assumption holds, then the Σ^y protocol is secure in the sesense of Definition 7.*

The security proof of Theorem 1 is given in Appendix A, which exploits the random oracle model and GDH assumption.

[4] e.g., the ephemeral public key generation subroutine, which takes the $\tilde{x}$ as input an outputs the ephemeral public key g^x.

Definition 9 (EphemeralSecrets-compromise Impersonation Attack). *We say that a KE-adversary $\mathcal{A}$ that has learned one ephemeral secret of a exposed session $s^u_{\hat{A}}$ established by a honest party $\hat{A}$ with other honest party $\hat{B}$,succeeds in a EphemeralSecrets-compromise impersonation (ECI) attacks as $\hat{A}$ against $\hat{B}$, if $\mathcal{A}$ is able to distinguish from random the session key of a session $s^v_{\hat{B}}$ at $\hat{B}$, for which the $\hat{A}$ and $\hat{B}$ both are uncorrupted, the session $s^v_{\hat{B}}$ is fresh and not matching to $s^u_{\hat{A}}$, and its execution status $\Lambda \in \{$pre-accept-key, accept-session$\}$.*

As discussed before, the protocol $\mathcal{P}$ and $HMQV$ are not resist with ECI attacks.

Lemma 3. *Under random oracle model, the Σ^y protocol provides resilience to the ephemeralsecrets-compromise impersonation attacks in the sense of Definition 9, under our proposed model.*

Proof. (sketch) In the Σ^y protocol those ephemeral secrets during key computing include $x, x + a\psi, x\psi + a, \sigma$. In our model, we allow the *KE-adversary* to learn those secrets via $SessionKeyReveal$ query, but with the restrictions that no more than one of them could be learned (otherwise it might lead the leakage of the long-term private key which is modeled by the query *Long-termKeyReveal*). Since those secrets are used to compute the *pre-sessionkey* σ are distinct among non-matching sessions, exposing one of them in session $s^u_{\hat{A}}$ would not affect other non-matching sessions.

5 Conclusion

In this paper we have discussed the deficiency of eCK and CK01 model. By contrast with security definitions in those two models, our proposed security model formalizes the authentication issue in a generic way when considering different type of AKE protocols. We have shown that the modeling of authentication is more fundamental, while designing and analyzing AKE protocol particularly involving strong adversary capabilities.

References

1. David Basin and Cas Cremers. From dolev-yao to strong adaptive corruption: Analyzing security in the presence of compromising adversaries. Cryptology ePrint Archive, Report 2009/079, 2009. `http://eprint.iacr.org/`.
2. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *STOC*, pages 419–428, 1998.
3. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, pages 139–155, 2000.
4. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.
5. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
6. Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: the three party case. In *STOC*, pages 57–66, 1995.
7. Luciano Bello. Debian open ssl predictable random number generator. Technical report, Debian.org, 2008.
8. Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In *IMA Int. Conf.*, pages 30–45, 1997.
9. Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the station-to-station (sts) protocol. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 154–170. Springer, 1999.
10. Colin Boyd, Yvonne Cliff, Juan Manuel González Nieto, and Kenneth G. Paterson. Efficient one-round key exchange in the standard model. In *ACISP*, pages 69–83, 2008.
11. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.

12. Ran Canetti and Hugo Krawczyk. Security analysis of ike's signature-based key-exchange protocol. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161. Springer, 2002.
13. Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 585–604. Springer, 2005.
14. Cas J. F. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the naxos authenticated key exchange protocol. In *ACNS*, pages 20–33, 2009.
15. Cas J.F. Cremers. Formally and practically relating the ck, ck-hmqv, and eck security models for authenticated key exchange. Cryptology ePrint Archive., August 2009.
16. T. Dierks and E. Rescorla. RFC 5246: The transport layer security (tls) protocol; version 1.2, August 2008.
17. Alan O. Freier, Philip Kariton, and Paul C. Kocher. The ssl protocol: Version 3.0. internet draft. 1996.
18. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
19. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
20. Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic compilers for authenticated key exchange. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 2010.
21. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.
22. Hugo Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, 2005.
23. Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.
24. Laurie Law, Alfred Menezes, Minghua Qu, Jerome A. Solinas, and Scott A. Vanstone. An efficient protocol for authenticated key agreement. *Des. Codes Cryptography*, 28(2):119–134, 2003.
25. Alfred Menezes. Another look at hmqv. Cryptology ePrint Archive, Report 2005/205, 2005. `http://eprint.iacr.org/`.
26. Alfred Menezes and Berkant Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. *IJACT*, 1(3):236–250, 2009.
27. Tatsuaki Okamoto. Authenticated key exchange and key encapsulation in the standard model. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 474–484. Springer, 2007.
28. Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2001.
29. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *EUROCRYPT*, pages 387–398, 1996.
30. Augustin P. Sarr, Philippe Elbaz-Vincent, and Jean-Claude Bajard. A new security model for authenticated key agreement. In *SCN*, pages 219–234, 2010.
31. Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.
32. Victor Shoup. On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012, 1999. `http://eprint.iacr.org/`.
33. Berkant Ustaoglu. Obtaining a secure and efficient key agreement protocol from (h)mqv and naxos. *Des. Codes Cryptography*, 46(3):329–342, 2008.
34. Berkant Ustaoglu. Comparing *sessionstatereveal* and *ephemeralkeyreveal* for diffie-hellman protocols. In *ProvSec*, pages 183–197, 2009.
35. Andrew C. Yao and Yunlei Zhao. A new family of practical non-malleable diffie-hellman protocols. Cryptology ePrint Archive, Report 2011/035, 2011. `http://eprint.iacr.org/`.

A Proof of Theorem 1

We first give a proof for the requirement that the session key is derived from the *pre-sessionkey*, i.e. the σ.

Lemma 4. *If H_1 and H_2 are modeled as random oracles, then the σ generated by each session of Σ^y protocol is pre-sessionkey.*

Proof. (sketch) It is trivially to see that the matching session would compute the same σ, so that we focus on its uniqueness among other non-matching sessions. We note that the exponents of σ (e.g. $x + a\psi, x\psi + a$) are derived from Schnorr's identification and signature scheme [31]. In which the x is chosen uniformly random for each session, and the ψ is a global unique value due to Lemma 2. By applying the proof methodology of Pointcheval-Stern [29] to our scheme, the σ is not only uniform random, but also distinct among non-matching sessions with overwhelming probability. Therefore, the σ in Σ^y is *pre-sessionkey* in sense of Definition 4.

We now verify that no polynomially bounded adversary can distinguish the session key of a fresh session from a randomly chosen session key. Let λ denote the security parameter, and let $\mathcal{A}$ be a polynomially (in λ) bounded adversary. The adversary $\mathcal{A}$ is said to be successful with non-negligible probability if $\mathcal{A}$ wins the distinguishing game with probability $1/2 + p(\lambda)$, where $p(\lambda)$ is non-negligible. The event that $\mathcal{A}$ is successful is denoted by M. Let the test session be $s := (\psi, \hat{A}, I, X, \hat{B}, R, Y)$ in which ψ is unique value between sessions and publicly known; and let H^* be the event that $\mathcal{A}$ queries H_3 with *pre-sessionkey* σ. Let $\overline{H^*}$ be the complement of event H^*, and let s^* be any completed session owned by an honest party such that $s^* \neq s$ and s^* is non-matching to s. Since s^* and s are distinct and non-matching, it can be seen that the inputs to the key derivation function H_3 are different for s and s^*. And, since H_3 is modeled as random oracle, it follows that $\mathcal{A}$ cannot obtain any information about the test session key from the session keys of non-matching sessions. Hence $\Pr(M \wedge \overline{H^*}) \leq 1/2$ and $\Pr(M) = \Pr(M \wedge H^*) + \Pr(M \wedge \overline{H^*}) \leq \Pr(M \wedge H^*) + 1/2$, whence $\Pr(M \wedge H^*) \geq p(\lambda)$. We will henceforth denote the event $M \wedge H^*$ by M^*. Assume further that $\mathcal{A}$ succeeds in an environment with n parties, activates at most t sessions within a party, makes at most h_1, h_2, h_3 queries to oracles H_1, H_2, H_3 respectively, and terminates after time at most $T_{\mathcal{A}}(\lambda)$. The following conventions will be used for the remainder of the security argument. The DDH oracle on input (g^a, g^b, g^c) returns the bit 1 if $g^{ab} = g^c$ and the bit 0 otherwise. Also, $\xi : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ is a random function known only to $\mathcal{S}$, such that $\xi(X, Y) = \xi(Y, X)$ for all $X, Y \in \mathbb{G}$. The algorithm $\mathcal{S}$, which simulates $\mathcal{A}$'s environment, will use $\xi(X, Y)$ to represent $CDH(X, Y)$ in situations where $\mathcal{S}$ may not know $\log_g^X$ and $\log_g^Y$. Except with negligible probability, $\mathcal{A}$ will not detect that $\xi(X, Y)$ is being used instead of $CDH(X, Y)$.

In addition, we define the parameter aux as constant for $SessionKeyReveal$ from 1 to 5 that correspond one-to-one with each secrets can be revealed by this query. With respect to the protocol Σ^y, those secretes for session $(\psi, \hat{A}, I, X, \hat{B}, R, Y)$ at party $\hat{A}$ including: $x, x + a\psi, x\psi + a, \sigma, k$.[5]. Since the uniqueness of $x + a\psi, x\psi + a$ and σ, the adversary is unable to establish two non-matching sessions to share the same those values. Therefore, we mainly focus on the leakage of x via $SessionKeyReveal$ in the proof, while no matching sessions existing.

Subsequently we use $\mathcal{A}$ to construct DLOG and CDH solver $\mathcal{S}$ respectively, that succeeds with non-negligible probability. We begin by considering the following complementary events:

E_1: There exists an honest party $\hat{A}$ such that $\mathcal{A}$, during its execution, queries H_1 with $(*, a)$ before or without issuing a *Long-termKeyReveal*$(\hat{A})$ query.

E_2: During the execution of every party $\hat{A}$, for which $\mathcal{A}$ first queries H_1 with $(*, a)$ after she issued *Long-termKeyReveal*$(\hat{A})$.

Since $\Pr(M^*)$ is non-negligible, it must be the case that either event $E_1 \wedge M^*$ or $E_2 \wedge M^*$ occurs with non-negligible probability. These possibilities are considered in §Case 1. and §Case 2. *Case* 1. Event $E_1 \wedge M^*$. We use $\mathcal{A}$ to construct an algorithm $\mathcal{S}$ that succeeds in solving the DLOG instance (V) with non-negligible probability provided that event $E_1 \wedge M^*$ occurs with non-negligible probability. $\mathcal{S}$ begins by establishing n parties. One of these parties, denoted $\hat{V}$, is selected at random and assigned the static public key V, and $v \in_R [1, q–1]$ is used to represent the

[5] For instance, the return of issuing $SessionKeyReveal(\hat{A}, sid, 2)$ is $x + a\psi$

corresponding static private key. The remaining parties are assigned random static key pairs. For each honest party $\hat{V}$, $\mathcal{S}$ maintains a list of at most t ephemeral key pairs, and two markers – a party marker and an adversary marker. The list is initially empty, and the markers initially point to the first entry of the list. Whenever $\hat{A}$ is activated to create a new session, $\mathcal{S}$ checks if the party marker points to an empty entry. If so then $\mathcal{S}$ selects a new ephemeral key pair on behalf of $\hat{A}$ as described in step 1 or 2 of the Σ^y protocol. If the list entry is not empty, then $\mathcal{S}$ uses the ephemeral key pair in that list entry for the newly created session. In either case the party marker is updated to point to the next list entry, and the adversary marker is also advanced if it points to an earlier entry. If $\mathcal{A}$ asks for an ephemeral public key query, then $\mathcal{S}$ selects a new ephemeral key pair on behalf of $\hat{A}$ as described in step step 1 or 2 of the protocol. $\mathcal{S}$ stores the key pair in the entry pointed to by the adversary marker, returns the public key as the query response, and advances the adversary marker. $\mathcal{S}$ activates the adversary $\mathcal{A}$ on this set of parties and awaits the actions of $\mathcal{A}$.

In addition to the above steps, $\mathcal{S}$ have to simulate the behavior of party $\hat{V}$ to generate corresponding ephemeral public key U without the knowledge of long-term secret v and maintains a separate list of at most t ephemeral key. The simulation function $Sim(\hat{V})$ defined as: while activating to create a new ephemeral key, the $\mathcal{S}$ creates a new simulation index i and selects a random ephemeral key $\tilde{u} \in_R \{0,1\}^\lambda$. Next, $\mathcal{S}$ selects $u \in Z_q$ independently and uniformly at random, except in the case of $\tilde{u}_i = \tilde{u}_{i'}$ for some previous session. For the exception, $\mathcal{S}$ sets $u_i = u_{i'}$. Then $\mathcal{S}$ presents $\mathcal{A}$ with the outgoing ephemeral public key $U = g^u$, finally the list contains $(i, \tilde{u}, u, U)$ and $\mathcal{S}$ returns U.

We next describe the actions of $\mathcal{S}$ in response to party activations and oracle queries.

1. $Send(\hat{A}, I, \hat{B}, R)$: $\mathcal{S}$ executes step 1 of the protocol, except if $\hat{A} = \hat{V}$ then $\mathcal{S}$ runs $Sim(\hat{A})$.
2. $Send(X, \hat{A}, I, \hat{B}, R)$ issued to $\hat{B} : \mathcal{S}$ executes step of the protocol. However, if $\hat{B} = \hat{V}$ then if $\hat{B} = \hat{V}$ then $\mathcal{S}$ runs $Sim(\hat{B})$, and sets $\theta_e = \xi(A, V)$ and $\sigma = \xi(X, UV^\psi)\xi(A, U^\psi V)$.
3. $Send(Y, \hat{A}, I, X, \hat{B}, R)$: $\mathcal{S}$ executes step 3 of the protocol. However, if $\hat{A} = \hat{V}$ then $\mathcal{S}$ sets $\theta_e = \xi(V, B)$ and $\sigma = \xi(Y, UV^\psi,)\xi(B, U^\psi V)$.
4. $H_1(*, z)$: For every new query, $\mathcal{S}$ computes $Z = g^z$. If $Z = V$, then $\mathcal{S}$ aborts and outputs $DLOG(V) = z$; otherwise $\mathcal{S}$ simulates a random oracle in the usual way.
5. $H_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way, by returning a random value from the range of H_2 for new queries and replaying answers if the queries were made before.
6. $H_3(\sigma, \psi)$:
   (a) If $\hat{A} = \hat{V}$ and $\sigma \neq \xi(Y, UV^\psi,)\xi(B, U^\psi V)$, then $\mathcal{S}$ sets $\tau = DDH(B, U^\psi V, \sigma(UV^\psi)^{-y})$.[6]
      i. If $\tau = 1$, then $\mathcal{S}$ returns $H_3(\xi(Y, UV^\psi)\xi(B, U^\psi V), \psi)$.
      ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
   (b) If $\hat{B} = \hat{V}$ and $\sigma \neq \xi(X, UV^\psi)\xi(A, U^\psi V)$, then $\mathcal{S}$ sets $\tau = DDH(A, U^\psi V, \sigma(UV^\psi)^{-x})$.
      i. If $\tau = 1$, then $\mathcal{S}$ returns $H_3(\xi(X, UV^\psi)\xi(A, U^\psi V), \psi)$.
      ii. If $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.
   (c) $\mathcal{S}$ simulates a random oracle in the usual way.
7. $SessionStateReveal(sid)$: $\mathcal{S}$ responds to the query faithfully.
8. $SessionKeyReveal(\hat{A}, sid, aux)$: $\mathcal{S}$ responds to the query faithfully.
9. $Long\text{-}termKeyReveal(\hat{A})$: If $\hat{A} = \hat{V}$, then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.
10. $EstablishParty(\hat{E}, E)$: $\mathcal{S}$ responds to the query faithfully.
11. $Corrupt(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.

[6] If the party $\hat{(V)}$ is communicate with herself (i.e., $\hat{A} = \hat{B} = \hat{V}$), then $\mathcal{S}$ have to use DDH oracle to tell whether σ is the result of CDH(or square-DH) without the knowledge of corresponding private key. Note that $\mathcal{S}$ process all ephemeral private key according to the simulation

12. $Test(sid)$: $\mathcal{S}$ responds to the query faithfully.
13. $\mathcal{A}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis. We observe that $\mathcal{S}$ perfectly simulates the $\mathcal{A}$'s environment until $\mathcal{A}$ queries *Long-termKeyReveal*$(\hat{V})$. Since in a perfect simulation $\mathcal{A}$ will query $H_1(*, DLOG(V))$ without first issuing a *Long-termKeyReveal*$(\hat{V})$ query. Hence, if event $E_1 \wedge M^*$ occurs with probability p_1, then $\mathcal{S}$ is successful with probability that is bounded by:

$$\Pr(\mathcal{S}) \geq \frac{p_1}{n}. \tag{1}$$

During the simulation, $\mathcal{S}$ performs group exponentiations, and simulates three random oracles. Let $q = \Theta(2^\lambda)$. Then a group exponentiation takes time $T_G = O(\lambda)$. Similarly, simulating oracles H_1, H_2 and H_3 take polynomially bounded time $T_{H_1}(\lambda), T_{H_2}(\lambda)$ and $T_{H_3}(\lambda)$, respectively. Therefore the running time $T_{\mathcal{S}}$ of $\mathcal{S}$ is bounded by

$$T_{\mathcal{S}} \leq (5T_G + 2T_{DDH} + T_{H_1} + T_{H_2} + T_{H_3})T_{\mathcal{A}}. \tag{2}$$

Case 2. Event $E_2 \wedge M*$. Let T_m be the event that: the test session has a matching session owned by an honest party. We further subdivide event $E_2 \wedge M^*$ into the following complementary events: (i) $E_{2a} = E_2 \wedge M^* \wedge T_m$ and (ii) $E_{2b} = E_2 \wedge M^* \wedge \overline{T_m}$. Let $p_2 = \Pr(E_2 \wedge M^*), p_{2a} = \Pr(E_{2a})$, and $p_{2b} = \Pr(E_{2b})$. Since E_{2a} and E_{2b} are complementary events we have $p_2 = p_{2a} + p_{2b}$. Therefore, if event $E_2 \wedge M^*$ occurs with non-negligible probability, then either E_{2a} or E_{2b} occurs with non-negligible probability. The two events are considered in §Case.2.1 and §Case.2.2.

*Case*2.1. In the event E_{2a}, we use $\mathcal{A}$ to construct an CDH solver $\mathcal{S}$ that succeeds in solving a CDH instance (U, V) with non-negligible probability, provided that event E_{2a} occurs with non-negligible probability. For initialization, $\mathcal{S}$ establishes n parties that are assigned random static key pairs. $\mathcal{S}$ randomly selects two parties $\hat{C}$, $\hat{D}$ and two integers $i, j \in_R [1, t]$ subject to the condition that $(\hat{C}, i) \neq (\hat{D}, j)$, note that party $\hat{C}$ might be equivalent to $\hat{D}$. And $\mathcal{S}$ selects ephemeral key pairs on behalf of honest parties as described in §Case 1, with the following exceptions. The ith ephemeral public key selected on behalf of $\hat{C}$ is chosen to be U and the corresponding ephemeral private key is $\tilde{u} \in_R \{0,1\}^\lambda$; note that $\mathcal{S}$ does not know $u = \log gU = H_1(\tilde{u}, c)$. Similarly, the jth ephemeral public key selected on behalf of $\hat{D}$ is V and the corresponding ephemeral private key is $\tilde{v} \in_R \{0,1\}^\lambda$; $\mathcal{S}$ does not know $v = \log gV = H_1(\tilde{v}, d)$. The sessions that are subsequently activated with ephemeral public keys U and V are denoted by s^U and s^V, respectively. Next we discuss the actions of $\mathcal{S}$ in response to party activations and oracle queries.

1. $Send(\hat{A}, I, \hat{B}, R)$: $\mathcal{S}$ executes step 1 of the protocol.
2. $Send(X, \hat{A}, I, \hat{B}, R)$ issued to $\hat{B}$: $\mathcal{S}$ executes step 2 of the protocol. If the session created is s^U or s^V, then $\mathcal{S}$ deviates from the protocol description by setting $\sigma = \xi(X, YB^\psi)\xi(A, Y^\psi B)$ where $Y \in \{U, V\}$ is the ephemeral public key of the created session.
3. $Send(Y, \hat{A}, I, X, \hat{B}, R)$: $\mathcal{S}$ executes step 3 of the protocol. If $X \in \{U, V\}$, then $\mathcal{S}$ deviates by not computing $x = H_1(\tilde{x}, a)$ in step 1a,3e, and by setting $\sigma = \xi(Y, XA^\psi)\xi(B, X^\psi A)$, where $X \in \{U, V\}$ is the ephemeral public key of the created session
4. $H_1(\tilde{x}, a)$: For every new query, $\mathcal{S}$ simulates a random oracle in the usual way except if $\tilde{x} = \tilde{u}$ and $\hat{A}$ (the party whose static public key is g^a) is the owner of s^U, or if $\tilde{x} = \tilde{v}$ and $\hat{A}$ is the owner of s^V , in which case $\mathcal{S}$ aborts with failure.
5. $H_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
6. $H_3(\sigma, \psi)$:
   (a) If $\{X, Y\} = \{U, V\}$, $\hat{A}$ and $\hat{B}$ are honest then $\mathcal{S}$ obtains $\tau = DDH(X, Y, \sigma Y^{-a\psi}(X^\psi A)^{-b})$. If $\tau = 1$ then $\mathcal{S}$ aborts $\mathcal{A}$ and is successful by outputting $CDH(U, V) = \sigma Y^{-a\psi}(X^\psi A)^{-b})$.

(b) If $X \in \{U, V\}$ and $\sigma \neq \xi(Y, XA^{\psi})\xi(B, X^{\psi}A)$, then $\mathcal{S}$ does the following. Using the DDH oracle, set $\tau = DDH(X, Y, \sigma(Y^{\psi}B)^{-a}X^{-b\psi})$.
   i. If $\tau = 1$ then, $\mathcal{S}$ returns $H_3(\xi(Y, XA^{\psi})\xi(B, X^{\psi}A), \psi)$.
   ii. item if $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

(c) If $Y \in \{U, V\}$ and $\sigma \neq \xi(X, YB^{\psi})\xi(A, Y^{\psi}B)$, then $\mathcal{S}$ does the following. Using the DDH oracle, set $\tau = DDH(X, Y, \sigma(X^{\psi}A)^{-b}Y^{-a\psi}) = 1$.
   i. If $\tau = 1$ then, $\mathcal{S}$ returns $H_3(\xi(X, YB^{\psi})\xi(A, Y^{\psi}B), \psi)$.
   ii. item if $\tau = 0$, then $\mathcal{S}$ simulates a random oracle in the usual way.

(d) $\mathcal{S}$ simulates a random oracle in the usual way.

7. *SessionStateReveal*(sid): $\mathcal{S}$ responds to the query faithfully.
8. *SessionKeyReveal*($\hat{A}, sid, aux$): $\mathcal{S}$ responds to the query faithfully.
9. *Long-termKeyReveal*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully.
10. *EstablishParty*($\hat{E}, E$): $\mathcal{S}$ responds to the query faithfully.
11. *Corrupt*($\hat{A}$): $\mathcal{S}$ responds to the query faithfully.
12. *Test*(sid): If the ephemeral public keys used in s are not U, V, then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.
13. $\mathcal{A}$ outputs a guess: $\mathcal{S}$ aborts with failure.

Analysis. $\mathcal{S}$'s simulation of $\mathcal{A}$'s environment is perfect except with negligible probability. The probability that $\mathcal{A}$ selects s^U or s^V as the test session and the other as its matching session is at least $\frac{2}{(nt)^2}$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in step 12), and suppose that event E_{2a} occurs. Let $\hat{A}$ and $\hat{B}$ denote the communicating parties for the test session and, without loss of generality, let $\hat{A}$ be the test session owner and U her ephemeral public key. Since $\tilde{u}$ is used only in the test session, $\mathcal{A}$ must obtain it via a *SessionStateReveal* query before making an H_1 query that includes $\tilde{u}$. Similarly, $\mathcal{A}$ must obtain $\tilde{v}$ from the matching session via a *SessionStateReveal* query before making an H_1 query that includes $\tilde{v}$. Under event E_2, the adversary first issues a $Long{-}termKeyReveal$ query to a party before making an H_1 query that includes that party's static private key. Since the test session is fresh, $\mathcal{A}$ can query for at most one value in each of the pairs $(\tilde{u}, a)$ and $(\tilde{v}, b)$; hence $\mathcal{S}$ does not abort as described in step 4. Under event M^*, except with negligible probability of guessing $\xi(U, V)$, $\mathcal{S}$ is successful as described in step 6a and does not fail as described in step 13. The success probability of $\mathcal{S}$ is bounded by

$$\Pr(\mathcal{S}) \leq \frac{2p_{2a}}{(nt)^2}. \tag{3}$$

We assume that a DDH oracle call takes polynomial time $T_{DDH}(\lambda)$ and the running time of $\mathcal{S}$ is bounded by

$$T_{\mathcal{S}} \leq (8T_G + 2T_{DDH} + T_{H_1} + T_{H_2} + T_{H_3})T_{\mathcal{A}}. \tag{4}$$

Case 2.2. For Event $E_{2b} \wedge M^*$, we use $\mathcal{A}$ to construct an algorithm $\mathcal{S}$ that succeeds in solving the CDH instance (U, YV^{ψ})[7] with non-negligible probability, provided that event $E_{2b} \wedge M*$ occurs with non-negligible probability. $\mathcal{S}$ establishes n parties, in which the party denoted by $\hat{V}$ is selected at random and assigned the static public key V, and $v \in_R [1, q–1]$ is used to represent the corresponding long-term private key. The remaining parties are assigned random static key pairs. Furthermore, $\mathcal{S}$ randomly selects a party $\hat{C}$ and integer $i \in_R [1, t[$. $\mathcal{S}$ selects ephemeral key pairs on behalf of honest parties as described in §Case.1, with the following exception. The *ith* ephemeral public key selected on behalf of $\hat{C}$ is chosen to be U and the corresponding ephemeral private key is $\tilde{u} \in_R \{0, 1\}^{\lambda}$;

[7] For the CDH instance (U, YV^{ψ}), in which the valuable Y might chosen by the adversary during the protocol run and it might be not known by $\mathcal{S}$.

note that $\mathcal{S}$ does not know $u = \log gU = H_1(\tilde{u}, c)$. The session that is subsequently activated with ephemeral public key U is denoted by s^U. Next we discuss the actions of $\mathcal{S}$ in response to party activations and oracle queries.

1. $Send(\hat{A}, I, \hat{B}, R)$: $\mathcal{S}$ executes step 1 of the protocol.
2. $Send(X, \hat{A}, I, \hat{B}, R)$ issued to $\hat{B}$: $\mathcal{S}$ executes step 2 of the protocol. Except if $\hat{B} = \hat{V}$, then $\mathcal{S}$ runs $Sim(\hat{B})$ and deviates from the protocol description by setting $\sigma = \xi(X, YV^{\psi})\xi(A, Y^{\psi}V)$ and $\theta_e = \xi(A, V)$.
3. $Send(Y, \hat{A}, I, X, \hat{B}, R)$: $\mathcal{S}$ executes step 3 of the protocol. Except if $\hat{A} = \hat{V}$ then $\mathcal{S}$ runs $Sim(\hat{B})$ and deviates *pre-sessionkey* by setting $\sigma = \xi(Y, XV^{\psi})\xi(B, X^{\psi}V)$, and $\theta_e = \xi(V, B)$.
4. $H_1(tilde(x), a)$: For every new query, $\mathcal{S}$ simulates a random oracle in the usual way except if $\tilde{x} = \tilde{u}$ and $\hat{A}$ (the party whose static public key is g^a) is the owner of s^U , in which case $\mathcal{S}$ aborts with failure.
5. $H_2(*)$: $\mathcal{S}$ simulates a random oracle in the usual way.
6. $H_3(\sigma, \psi)$:
   (a) If $\hat{A} = \hat{V} \neq \hat{B}$ and $\sigma \neq \xi(Y, XV^{\psi})\xi(B, X^{\psi}V)$, then $\mathcal{S}$ obtains $\tau = DDH(Y, XV^{\psi}, \sigma(X^{\psi}V)^{-b})$.
      i. If $\tau = 1$ and $Y = U$, then $\mathcal{S}$ aborts $\mathcal{A}$ and outputs $CDH(U, XV^{\psi}) = \sigma(X^{\psi}V)^{-b}$
      ii. If $\tau = 1$ and $Y \neq U$, then $\mathcal{S}$ returns $H_3(\xi(Y, XV^{\psi})\xi(B, X^{\psi}V), \psi)$.
      iii. If $\tau = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (b) If $\hat{B} = \hat{V} \neq \hat{A}$ and $\sigma \neq \xi(X, YV^{\psi})\xi(A, Y^{\psi}V)$, then $\mathcal{S}$ obtains $\tau = DDH(X, YV^{\psi}, \sigma(Y^{\psi}V)^{-a})$.
      i. If $\tau = 1$ and $X = U$, then $\mathcal{S}$ aborts $\mathcal{A}$ and outputs $CDH(U, YV^{\psi}) = \sigma(Y^{\psi}V)^{-a}$
      ii. If $\tau = 1$ and $Y \neq U$, then $\mathcal{S}$ returns $H_3(\xi(X, YV^{\psi})\xi(A, Y^{\psi}V), \psi)$.
      iii. If $\tau = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (c) If $\hat{A} = \hat{B} = \hat{V}$.
      i. Else if $Y = U$ and $\sigma \neq \xi(X, YV^{\psi})\xi(A, Y^{\psi}V)$ then $\mathcal{S}$ obtains $\tau = DDH(A, Y^{\psi}V, \sigma(YV^{\psi})^{-x})$.
         A. If $\tau = 1$ then $\mathcal{S}$ aborts $\mathcal{A}$ and outputs $CDH(V, U^{\psi}V) = \sigma(YV^{\psi})^{-x}$
         B. If $\tau = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
      ii. Else if $X = U$ and $\sigma \neq \xi(Y, XV^{\psi})\xi(B, X^{\psi}V)$ then $\mathcal{S}$ obtains $\tau = DDH(B, X^{\psi}V, \sigma(XV^{\psi})^{-y})$.
         A. If $\tau = 1$ then $\mathcal{S}$ aborts $\mathcal{A}$ and outputs $CDH(V, U^{\psi}V) = \sigma(XV^{\psi})^{-y}$
         B. If $\tau = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
      iii. If both $Y, X \neq U$ and $\sigma \neq \xi(X, YV^{\psi})\xi(A, Y^{\psi}V)$ then $\mathcal{S}$ obtains $\tau = DDH(A, Y^{\psi}V, \sigma(YV^{\psi})^{-x})$.
         A. If $\tau = 1$ then $\mathcal{S}$ returns $H_3(\xi(X, YV^{\psi})\xi(A, Y^{\psi}V), \psi)$.
         B. If $\tau = 0$, $\mathcal{S}$ simulates a random oracle in the usual way.
   (d) $\mathcal{S}$ simulates a random oracle in the usual way.
7. $SessionStateReveal(sid)$: $\mathcal{S}$ responds to the query faithfully.
8. $SessionKeyReveal(\hat{A}, sid, aux)$: $\mathcal{S}$ responds to the query faithfully.
9. $Long\text{-}termKeyReveal(\hat{A})$: $\mathcal{S}$ responds to the query faithfully, except if $\hat{A} = \hat{V}$ in which case $\mathcal{S}$ aborts with failure.
10. $EstablishParty(\hat{E}, E)$: $\mathcal{S}$ responds to the query faithfully.
11. $Corrupt(\hat{A})$: $\mathcal{S}$ responds to the query faithfully.
12. $Test(sid)$: If the peer of sid is not $\hat{V}$ or the outgoing ephemeral public key is not U then $\mathcal{S}$ aborts with failure. Otherwise $\mathcal{S}$ responds to the query faithfully.
13. $\mathcal{A}$ outputs a guess : $\mathcal{S}$ aborts with failure.

Analysis. The probability that the test session has peer $\hat{V}$ and outgoing ephemeral public key U is at least $\frac{1}{(n^2 t)}$. Suppose that this is indeed the case (so $\mathcal{S}$ does not abort in step 12), and suppose that event E_{2b} occurs. Let $\hat{A}$ be the owner of the test session s^U. The experiment may fail if $\mathcal{S}$ queries H_1 with $(\tilde{u}, a)$. Since $\tilde{u}$ is used only in the test session, $\mathcal{S}$ must obtain it via an *SessionStateReveal*

query before making an H_1 query that includes $\tilde{u}$. Under event E_2, the adversary first issues a *Long-KeyReveal* query to a party before making an H_1 query that includes that party's static private key. Since the test session is fresh, $\mathcal{A}$ can query for at most one value in the pair $(\tilde{u}, a)$; hence $\mathcal{S}$ does not abort as described in step 4. Since the test session is fresh, $\mathcal{A}$ is not allowed to issue a *Long-KeyReveal* query to $\hat{V}$; hence $\mathcal{S}$ does not abort as described in step 9. In event $\overline{T_m}$ the test session has no matching session, $\mathcal{A}$ must have obtained the $CDH(U, YV^{\psi})$ to generate the σ'. In this case, $\mathcal{S}$ is successful as described in step 6 and does not abort in step 13. The success probability of $\mathcal{S}$ is bounded by

$$\Pr(\mathcal{S}) \leq \frac{p_{2b}}{(n^2 t)}. \tag{5}$$

The running time of $\mathcal{S}$ is bounded by

$$T_{\mathcal{S}} \leq (5T_G + 2T_{DDH} + T_{H_1} + T_{H_2} + T_{H_3})T_{\mathcal{A}}. \tag{6}$$

OverallAnalysis. Combining the results from §Case 1, §Case 2.1 and §Case 2.2, we conclude that for every adversary $\mathcal{A}$ there is an algorithm $\mathcal{S}$ that solves DLOG and the GDH instance with running time $T_{\mathcal{S}}$ and success probability $Pr(\mathcal{S})$, where

$$Pr(\mathcal{S}) \geq max(\frac{p_1}{(n)}, \frac{2p_{2a}}{(nt)^2}, \frac{p_{2b}}{(n^2 t)}) \tag{7}$$

and

$$T_{\mathcal{S}} \leq (8T_G + 2T_{DDH} + T_{H_1} + T_{H_2} + T_{H_3})T_{\mathcal{A}}. \tag{8}$$

B A variant of MQV protocol

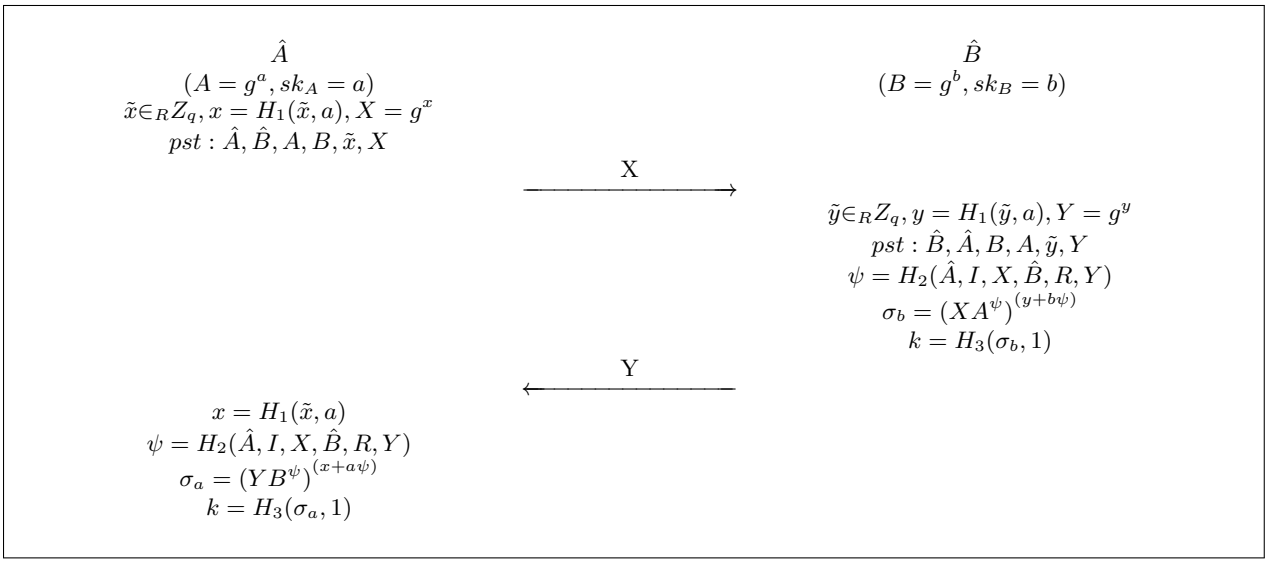


Fig. 2: V-MQV Protocol under the model in section 3.

C Implication relation of the security definitions

We denote our proposed security model by SAC (which is the abbreviation of 'strong adaptive corruption'). In this sub-section, we formally relate the CK01, eCK and SAC security models for AKE protocols following the method used in [13].

Let the advantage of some PPT adversary, $\mathcal{A}_{01}$, in the CK01 model be $Adv^{\mathcal{A}_{ck01}}$, the advantage of some PPT adversary, $\mathcal{A}_{eck}$, in the eCK model be $Adv^{\mathcal{A}_{eck}}$, and the advantage of some PPT adversary, $\mathcal{A}_{sac}$, in the SAC model be $Adv^{\mathcal{A}_{sac}}$. Moreover, for clear demonstration, we differentiate the $SessionStateReveal^{sac}$ query in SAC model from the $SessionStateReveal^{ck01}$ in CK01 model.

Lemma 5. *For any protocol proven secure in SAC model implies security in eCK model, in the same security parameter.*

Proof. Please note that the one of the major differences among SAC, eCK and CK01 models are variant of CK01 model models are the adversary capabilities. In particular, the SAC and eCK models both allow adversary to reveal part of the state of the test session. We claim that the $SessionStateReveal^{sac}$ in SAC model is at least as strong as the *EphemeralKeyReveal* in eCK model.

We construct an adversary $\mathcal{A}_{sac}$ trying to break the security of a KE protocol π in the SAC model using an adversary $\mathcal{A}_{eck}$ against the security of the same KE protocol in the eCK model. In the simulation, the $\mathcal{A}_{sac}$ reply the queries asked by $\mathcal{A}_{eck}$ with its own relative oracles and forwards the received answers to $\mathcal{A}_{eck}$. The $\mathcal{A}_{sac}$ in particular responds to the *EphemeralKeyReveal* query using $SessionStateReveal^{sac}$. Eventually, $\mathcal{A}_{eck}$ outputs a guess bit b^{eck} and $\mathcal{A}_{sac}$ will output the same bit as its own guess answer. The $\mathcal{A}_{sac}$ wins the experiment if $\mathcal{A}_{eck}$ does. As discussed in section 2.3, the revelation of ephemeral secrets by *EphemeralKeyReveal* are comprised by those public states modeled in SAC model, which are contained in the results of $SessionStateReveal^{sac}$ (i.e. the $SessionStateReveal^{sac}$ can be used as a substitution of *EhpemeralKeyReveal* for $\mathcal{A}_{eck}$). Hence, the $\mathcal{A}_{sac}$ is able to simulate exactly the view of $\mathcal{A}_{eck}$.

The matching sessions in eCK and SAC model are both defined by the session identifier which is concatenated by messages exchanged. Therefore, the matching sessions in eCK are also matching in SAC model. We re-examine the difference concerning the definition of session freshness in both models, is the use of $EphemeralKeyReveal$ in eCK model substituted by $SessionStateReveal^{sac}$ in SAC model. So that the fresh session in eCK model is also fresh in SAC model.

In order to analyze $Adv^{\mathcal{A}_{eck}}$, we first consider the case that SAC's *Test* oracle returns a random key, in whch the probability of $\mathcal{A}_{eck}$ guessing the correct b^{eck} is $\frac{1}{2}$ since it cannot gain any information about the hidden bit b^{sac}. On the second, we take into account the case that the SAC's *Test* oracle returns the real session key. In which case, we claim that the $Adv_{\pi}^{\mathcal{A}_{sac}}$ is bound to $Adv_{\pi}^{\mathcal{A}_{eck}}$, due to the $\mathcal{A}_{sac}$ perfectly simulating the environment of $\mathcal{A}_{eck}$. In other words, there doesn't exist protocol which is proven secure in SAC model but not in eCK model.

Lemma 6. *For any protocol proven secure in SAC model implies security in CK01 model, in the same security parameter.*

Proof. The proof follows Lemma 5. We first construct an adversary $\mathcal{A}_{sac}$ trying to break the security of a KE protocol π in the SAC model using an adversary $\mathcal{A}_{ck01}$ against the security of the same KE protocol in the eCK model. Both CK01 and SAC adversary are allowed to reveal the session states of sessions. The CK01 model doesn't specify what kinds of session states can be revealed. In contrast, we split session states into public states and local states of each secure subroutine (e.g., signature and encryption) in SAC model. We treat those secure subroutines as black box

(except for the key generation procedure which interests the adversary most) and only consider the revelation of their inputs and outputs, because these local states are highly rely on specific security of corresponding cryptographic primitive other than the AKE protocol design. Note that if the subroutines are nested then we only reveal the inputs and outputs of outer most subroutine. Hence, we believe that $SessionStateReveal^{sac}$ and *SessionKeyReveal* in SAC model have already covered the most important session states. In the simulation, the $\mathcal{A}_{sac}$ replies the queries asked by $\mathcal{A}_{eck}$ with its own relative oracles and forwards the received answers to $\mathcal{A}_{eck}$. Meanwhile, we assume that the $\mathcal{A}_{ck01}$ only ask $SessionStateReveal^{ck01}$ query once. When $\mathcal{A}_{ck01}$ querying $SessionStateReveal^{ck01}$, the $\mathcal{A}_{sac}$ forwards the answers from $SessionStateReveal^{sac}$ and *SessionKeyReveal*. Moreover, the fresh sessions in CK01 model remain fresh in SAC model. Since the matching sessions in SAC and CK01 model are both defined by the unique session identifier, hence we can conclude that all CK01 partners are also SAC partners. Thereafter, the $\mathcal{A}_{sac}$ is able to simulate exactly the view of $\mathcal{A}_{ck01}$.

In order to analyze $Adv^{\mathcal{A}_{eck}}$, we first consider the case that SAC's *Test* oracle returns a random key, in which the probability of $\mathcal{A}_{ck01}$ guessing the correct b^{ck01} is $\frac{1}{2}$ since it cannot gain any information about the hidden bit b^{sac}. On the second, we take into account the case that the SAC's *Test* oracle returns the real session key. Due to the $\mathcal{A}_{sac}$ perfectly simulating the environment of $\mathcal{A}_{eck}$, if $Adv_{\pi}^{\mathcal{A}_{ck01}}$ is non-negligible advantage, so does $Adv_{\pi}^{\mathcal{A}_{sac}}$ in violation of our security assumption of π.

Discussion on non-implication relations. Although the SAC security implies the security in eCK and CK model, it is trivially to see that the reverse implications do not hold. On the one hand, as the definition of session identifier and matching sessions in eCK model, the eCK partnership cannot support symmetric role application other than SAC model. And as discussion in section 2.3, there exist protocols which are proven secure in eCK model but not resistant with the leakage of session key related secrets; more precisely, the protocols without generating *pre-sessionkey* are not secure in SAC model. On the other hand, the CK01 model doesn't allow for *SessionStateReveal* queries on the test session and its matching session. And the compromise of the long-term private key of the participant that executes the test session before the test session expires is not allowed in the CK01 model. As a result, in contrast with SAC model, the CK01 model is not able to resist with the leakage of session states on test session and detect key compromise impersonation (KCI) attacks.