A preliminary version of this paper appears in Advances in Cryptology - ASIACRYPT 2025. This is the full version.

Universally Composable Subversion-Resilient Authenticated Key Exchange *

Jiahao Liu †
liujiahao14@nudt.edu.cn

Yi Wang ‡
wangyi14@nudt.edu.cn

Rongmao Chen✉ §
chromao@nudt.edu.cn

Xinyi Huang ¶
xyhuang81@gmail.com

Jinshu Su ‖
sjs@nudt.edu.cn

Moti Yung **
motiyung@gmail.com

August 25, 2025

Abstract

Subversion-resilient cryptography has garnered increasing attention in recent years due to growing concerns about cryptographic subversions in real-world applications. Among the existing countermeasures, the notion of cryptographic reverse firewalls (RFs), initially proposed by Mironov and Stephens-Davidowitz (EUROCRYPT 2015) and later extended by Chakraborty et al. (EUROCRYPT 2022) to the universally composable (UC) model, has proven to be a powerful tool for building subversion-resilient cryptographic protocols. In this work, we focus on designing subversion-resilient authenticated key exchange (AKE) protocols, which are critical components of secure Internet communication. We present the first generic framework for subversion-resilient UC-secure AKE protocols leveraging RFs. Inspired by the state-of-the-art advancements by Chakraborty et al. (ASIACRYPT 2024), we address subversions: where a party's implementation is covertly altered to exfiltrate secrets or behave unpredictably when triggered by adversarial inputs. A key contribution of our work is the introduction of a new AKE functionality which, for the first time, incorporates security against key control, an essential aspect of achieving subversion resilience. We also provide a concrete instantiation of our framework, demonstrating its feasibility in practice. Notably, the RFs in our proposed AKE protocol are **transparent**, an important property of RF as defined originally, which allows deployment of RF without all parties explicitly knowing about it and allows robust security. Achieving transparency for RFs has been widely regarded as challenging, particularly when addressing broader subversion attacks (e.g., input-trigger attacks) in the UC model. Our approach, thus, not only advances the state of AKE protocol design, but also offers insights into building other subversion-resilient protocols in the UC model using transparent RFs.

*We would like to thank anonymous reviewers of Eurocrypt 2025, Crypto 2025 and Asiacrypt 2025 for their helpful comments and suggestions. This work is supported in part by the National Natural Science Foundation of China (Grant No.62032005, No.62425205, No.62122092, No.62202485 and No.62372462) and the Young Elite Scientists Sponsorship Program by China Association for Science and Technology (No.YESS20230028).

†College of Computer Science and Technology, National University of Defense Technology, Changsha, China
‡College of Computer Science and Technology, National University of Defense Technology, Changsha, China
§College of Computer Science and Technology, National University of Defense Technology, Changsha, China
¶Jinan University, Guangzhou, China
‖Academy of Military Science, Beijing, China
**Google LLC & Columbia University, New York, USA

Contents

1 Introduction **2**
1.1 Our Contributions . . . 3

2 Results Overview and Related Works **4**
2.1 AKE Functionality . . . 4
2.2 Our Technology Roadmap . . . 4
2.3 Related Work . . . 8

3 Definitions and Components **9**
3.1 Key Encapsulation Mechanism . . . 9
3.2 Commitment Scheme . . . 11

4 The srUC Model **12**
4.1 Corruption Cases . . . 13
4.2 Wrapped Subversion-Resilient UC Security . . . 14
4.3 Properties of Reverse Firewalls . . . 14

5 Ideal Functionality for AKE **15**

6 Generic Construction **18**
6.1 Building Blocks . . . 18
6.2 Protocol srAKE . . . 18

7 Instantiations **24**

A Basic Definitions **31**
A.1 Diffie-Hellman Assumptions . . . 31
A.2 Indistinguishable Distribution Ensembles . . . 31
A.3 Properties for Commitment Scheme . . . 32

B Definitions in the srUC Model [24] **32**

C Proof of Lemma 1 **33**

D Omitted Description and Security Analysis of srAKE **34**
D.1 A Textual Description of srAKE . . . 34
D.2 Correctness Analysis . . . 34
D.3 Proof of Lemma 4 . . . 36
D.4 Proof of Lemma 5 . . . 37

E Omitted Instantiations **41**
E.1 Overview of Instantiations . . . 41
E.2 Security Analysis of Instantiations . . . 41
E.2.1 The Security of Π_{Com} and Π'_{Com} . . . 41
E.2.2 The Security of DHK . . . 44
E.2.3 The Security of ElGamal-KEM . . . 44

1 Introduction

Authenticated Key Exchange (AKE) protocols enable two parties to establish a session key securely over an insecure communication channel and mutually authenticate the parties' identities. Hence these protocols are essential for secure communications in various applications, including secure messaging, online transactions, and establishing secure connections in networks (i.e., TLS/SSL).

Traditional security models for AKE protocols implicitly assume that protocol implementations strictly adhere to the specifications, and do not consider such a powerful adversary that subverts the honest implementation to embed cryptographic trapdoors so that it can retrieve sensitive information covertly, thus undermining the security of the AKE protocols. These cryptographic subversion attacks, identified as kleptographic attacks by Young and Yung [58, 59] in the 1990s, initially regarded somewhat theoretical, until Snowden's revelations in 2013 exposed the potential usage of such attack in mass surveillance by intelligence agencies [50], which rekindled the attention of the cryptographic community [9, 30, 8, 13, 25, 2, 5, 36, 45, 57]. In another investigation, real-world AKE protocols, such as the TLS handshake, have been shown to be vulnerable to subversion attacks [14].

Subversion-Resilient UC Model. Note that an AKE protocol is often integrated as a submodule of complex protocols such as TLS [33], Signal [28, 17] and WireGuard [42], where multiple instances run concurrently, rather than being used in a standalone fashion. It is, therefore, highly desirable to construct AKE protocols that offer composable security when considering the presence of subversion attacks.

The study of universally composable (UC) subversion-resistant cryptography was initiated by Chakraborty et al. [24]. They extended the notion of cryptographic reverse firewall (RF) proposed by Mironov and Stephens-Davidowitz [49], which has been widely adopted to defend against subversion attacks for various primitives or protocols [5, 19, 26, 32, 20, 21, 15, 23], to the setting of the UC model (hereafter referred to as the srUC model). In this model, each party consists of a core and an RF. The core holds the secret inputs of the party and generates the messages of the protocol, while the RF does not have access to the secrets of the core and sanitizes the incoming/outgoing messages for the core with randomness.

At ASIACRYPT 2024, Chakraborty et al. [23] extended the srUC model to the unauthenticated setting, and constructed the first password-authenticated key exchange (PAKE) protocol under this model. The core idea is to first realize a subversion-resilient PAKE protocol assuming the existence of an ideal functionality $\mathcal{F}_{\mathsf{SAT}}$ for sanitizable authenticated communication, and then to transform it into a new protocol over unauthenticated channels via "split functionalities" [6]. In particular, they proposed a protocol that realizes the split version of $\mathcal{F}_{\mathsf{SAT}}$ in the unauthenticated setting and is an essential component of the transformation. In the message authentication phase of this protocol, the RF must ask the core to generate a signature for the sanitized message. That is, the RF in this PAKE protocol is not transparent, and the core is aware of the existence of its RF.

Transparency of RFs. It is worth mentioning that in the original proposal of RFs [49], Mironov and Stephens-Davidowitz stressed that they were interested in RFs that preserve the functionality of an already working protocol, and the proposed RFs are **all transparent** in the sense that the behavior of a party composed of an honest core and an RF is identical to the behavior of that honest core. Since a transparent RF does not change the behavior of the core, it could be deployed as a security enhancement module on the fly and without drastic changes throughout the overall system. Thus, it is highly preferable to design subversion-resilient protocols with transparent RFs in practice if we hope to actually employ the mechanism.

The above observation aligns with a recent work by Arnold et al. [3], which argues that non-transparent RFs, if corrupted, would drastically weaken the security guarantee provided by the srUC model. To address this, they proposed a simpler model that captures RFs within the *plain* UC model, enabling the reuse of existing tools (e.g., the composition theorem) and UC-secure primitives. They proved that to establish the UC security of a protocol against subversion attacks under their model, it suffices to demonstrate that the underlying RFs satisfy certain properties, including transparency. However, the subversion attacks considered in their model are restricted to subverting the randomness used by the cores. Other types of subversion attacks, particularly input-trigger attacks—where a subverted core behaves in an unexpected way upon receiving a triggering message crafted by the adversary—fall outside the scope of Arnold et al.'s model. Note that these attacks have been thoroughly examined in the design of subversion-resilient PAKE protocols in the srUC model [23].

Motivated by the aforementioned observations and driven by the goal of designing subversion-resilient AKE protocols, we pose the following question, left open by prior works, as the central focus of this work:

Is it possible to design an AKE protocol with transparent RFs in the srUC *model?*

It is worth noting that AKE protocols, like PAKE protocols, also run over unauthenticated channels. Hence, a key challenge in addressing the aforementioned question lies in mitigating the subversion attacks captured by the srUC model in the unauthenticated setting, as studied by Chakraborty et al. [23], while leveraging transparent RFs — a goal that was not achieved in prior work.

1.1 Our Contributions

This paper provides a positive answer to the above question by constructing the first subversion-resilient AKE protocol srAKE with transparent RFs under the srUC model [24] (in the setting of unauthenticated links). In particular, we show a generic framework built on key encapsulation mechanisms (KEMs) and commitment schemes. The starting point of our construction is a highly efficient key exchange protocol by Cohn-Gordon et al. [29]. In the next section, we give a high-level explanation on how this protocol can be modified to meet the requirements of **transparent RFs** and to **defend against input-trigger attacks**. To demonstrate the feasibility of the proposed framework, we provide a concrete instantiation of the underlying KEMs and commitment schemes.

We also present a variant of the AKE functionality by Jarecki et al. [43] that, for the first time, captures *security against key control*. This property requires that no party (even a corrupted or subverted one) can have any control over the session key. Recall that the goal of subversion-resilience is to prevent the leakage of sensitive information and to maintain the security in the presence of subversion attacks. It seems that security against key control does not contribute to subversion-resilience. However, when security against key control is not guaranteed, it is possible for a subverted party to control, say, certain bits of the session key as requested by the subversion attacker, and the attacker would obtain the key information. Thus, Dodis et al. [32] emphasized that security against key control is essential for a key-agreement protocol to be subversion-resilient.

Table 1: Security properties captured by ideal functionalities for AKE.

	[18, 41]	[43]	[39, 54]	**Ours**
Forward Secrecy	full	full	weak	weak
KCI Security	×	✓	✓	✓
Security against Key Control	×	×	×	✓

2 Results Overview and Related Works

2.1 AKE Functionality

Before designing UC-secure AKE protocols, it is necessary to determine the target ideal functionality, as there are a number of ideal functionalities for AKE [18, 41, 43, 39, 54] to date. Table 1 shows the comparison of these functionalities in terms of security properties. One can note that none of them imply security against key control. Our new ideal functionality for AKE is adapted from $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ [43] which extends the standard key exchange ideal functionality [18] with resistance to Key-Compromise Impersonation (KCI) attacks (or KCI security for short).

Security against Key Control. In $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$, the adversary is allowed to control the session key of an uncompleted session between parties P and P', denoted by $\langle ssid, \mathsf{P}, \mathsf{P}' \rangle$, when (i) $\langle ssid, \mathsf{P}, \mathsf{P}' \rangle$ is Compromised, or (ii) P or P' is corrupted. In particular, a fresh session $\langle ssid, \mathsf{P}, \mathsf{P}' \rangle$ is marked Compromised when the adversary takes an Impersonate action on a Compromised P'. The Compromised action on P' models the leakage of P''s secret keys to the adversary. In contrast to a Compromised party, a corrupted party is controlled by the adversary. Since both an adversary that holds P''s secret keys and impersonates P' in session $\langle ssid, \mathsf{P}, \mathsf{P}' \rangle$ and a corrupted party could control the session key, $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ does not imply security against key control.

To capture security against key control, our AKE functionality only permits the adversary to control the session key when (i) a party is corrupted and the adversary impersonates (i.e., takes an Impersonate action on) this party's counterpart, or (ii) both parties are corrupted, as in either case the adversary is able to obtain the session key before the end of the session.

Weak Forward Secrecy. $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ captures full forward secrecy in the sense that once a session $\langle ssid, \mathsf{P}, \mathsf{P}' \rangle$ is completed, the adversary can no longer obtain the session key by compromising the party P or P'. Weak forward secrecy is the same as full forward secrecy except that the adversary is passive during session execution. A standard way to build AKE protocols with full forward secrecy is to add key confirmation messages derived from the session key to an AKE protocol with weak forward secrecy. For example, the key confirmation messages in the HMQV-C protocol [44] are two MAC values (computed on 0 and 1) under the session key. Since the session key is unknown to the RFs, it is unlikely for the RFs to check and sanitize these messages to defend against subversion attacks. Therefore, we only consider weak forward secrecy.

2.2 Our Technology Roadmap

Our starting point is the implicitly authenticated key exchange protocol Π by Cohn-Gordon et al. [29]. Assume that the public parameters of Π include $g \in \mathbb{G}$, where $\mathbb{G}$ is a cyclic

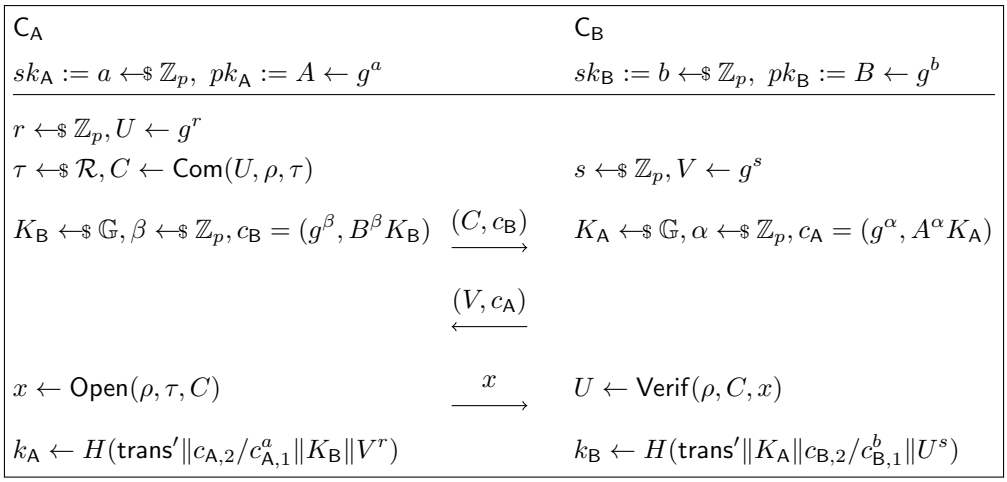


Figure 1: The protocol Π' where $\mathsf{C_A}$ first sends the commitment of U and then opens it after receiving V from $\mathsf{C_B}$. Let $g \in \mathbb{G}, p = |\mathbb{G}|$ be public parameters. Let $\Pi_{\mathsf{com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$ be the commitment scheme. $\rho \leftarrow\$ \mathsf{KGen}(1^\lambda)$ represents the public parameters of Π_{com}, and $\mathcal{R}$ is the randomness space of Π_{com}.

group of prime order p. In Π, two parties $\mathsf{C_A}$, $\mathsf{C_B}$ with long-term private/public key pairs $(a, A = g^a)$ and $(b, B = g^b)$ send $U = g^r$ and $V = g^s$ to each other, and the session key $k_\mathsf{A} = H(\mathsf{trans} \| V^a \| B^r \| V^r) = H(\mathsf{trans} \| A^s \| U^b \| U^s) = k_\mathsf{B}$ where H is a hash function and trans is a concatenation of the transcript (U, V), party identities $(\mathsf{C_A}, \mathsf{C_B})$ and long-term public keys (A, B). In particular, V^r, U^s are ephemeral-ephemeral DH values, and V^a, A^s, B^r, U^b are static-ephemeral DH values.

Modifying Static-Ephemeral DH Values. Note that the computation of the DH values V^a, U^b can be seen as a decapsulation of the ciphertexts V and U with the long-term secret keys a, b. It is reasonable to make the following changes to the protocol Π while maintaining the functionality. $\mathsf{C_A}(\mathsf{C_B})$ sends an extra ElGamal encryption $c_\mathsf{B} = (g^\beta, B^\beta K_\mathsf{B})(c_\mathsf{A} = (g^\alpha, A^\alpha K_\mathsf{A}))$ of a random key $K_\mathsf{B}(K_\mathsf{A})$ under the long-term public key $B(A)$ to each other, and the session key $k_\mathsf{A} = H(\mathsf{trans}' \| c_{\mathsf{A},2}/c_{\mathsf{A},1}^a \| K_\mathsf{B} \| V^r) = H(\mathsf{trans}' \| K_\mathsf{A} \| c_{\mathsf{B},2}/c_{\mathsf{B},1}^b \| U^s) = k_\mathsf{B}$. The transcript in trans' is $(U, c_\mathsf{B}, V, c_\mathsf{A})$. We remark that this modification ensures that U and V are only used for computing the ephemeral-ephemeral DH value, making it easier to extract the generic framework later.

Achieving Security against Key Control. Obviously, this modified protocol does not provide security against key control. After receiving (U, c_B) from $\mathsf{C_A}$, $\mathsf{C_B}$ can control the session key k by repeatedly sampling K_A or s until k meets certain requirements. To achieve security against key control, we follow the approach of Dodis et al. [32] and obtain a new protocol Π' shown in Fig. 1. That is, $\mathsf{C_A}$ commits to U and sends only the commitment of U and c_B to $\mathsf{C_B}$. Upon receiving (V, c_A) from $\mathsf{C_B}$, $\mathsf{C_A}$ opens the commitment to $\mathsf{C_B}$. In this case, $\mathsf{C_B}$ cannot derive the session key k until it receives the opening from $\mathsf{C_A}$, and loses control of k.

In Section 7, the commitment scheme depicted in Fig. 1 is instantiated as follows. The commitment to U is defined as $C = (g^y, h^y U)$, where $h \in \mathbb{G}$ is a public parameter and $y \leftarrow\$ \mathbb{Z}_p$. The opening of the commitment C is a random value y^*. To verify the opening, one first parses $C = (C_1, C_2)$, checks whether $C_1 = g^y$ and then either outputs C_2/h^y or $\perp$, depending on the validity of C.

The First Attempt: Adding Reverse Firewalls. Next we consider inserting RFs for $\mathsf{C_A}$ and

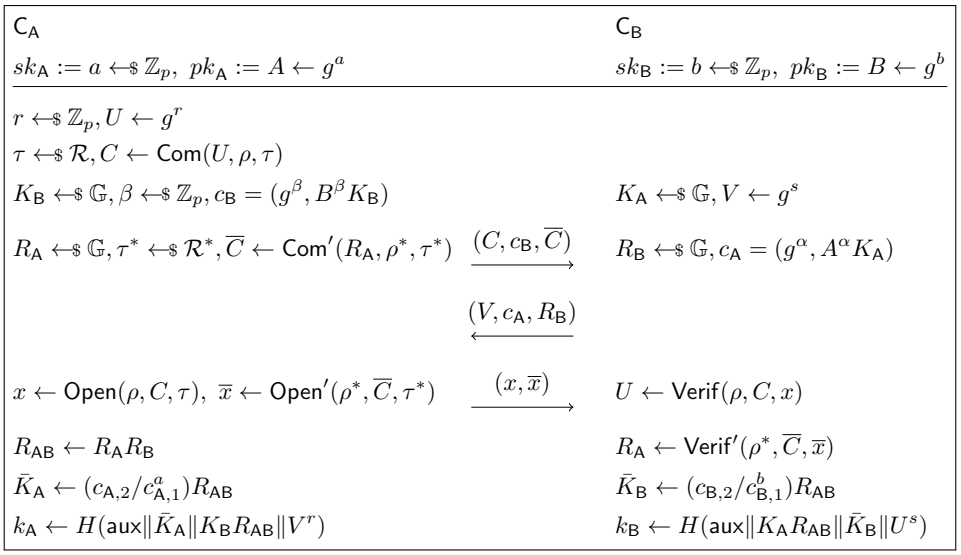


Figure 2: The protocol Π'' is the same as Π' except that 1) $\mathsf{C_A}$ and $\mathsf{C_B}$ perform an extra coin tossing protocol, 2) the derivation of the session key involves the negotiated element R_AB, and 3) aux is identical to trans' but excludes the protocol transcript. Let $\Pi'_\mathsf{Com} = (\mathsf{KGen}', \mathsf{Com}', \mathsf{Open}', \mathsf{Verif}')$ be another commitment scheme, with $\rho^* \leftarrow\!\$\ \mathsf{KGen}'(1^\lambda)$ as its public parameters and $\mathcal{R}^*$ as its randomness space.

$\mathsf{C_B}$ to defend against subversion attacks. In particular, the RF $\mathsf{F_A}$ for $\mathsf{C_A}$ must rerandomize the outgoing ciphertext c_B and maul its underlying key K_B to K'_B using randomness Δ_B. Assume that the RF $\mathsf{F_B}$ for $\mathsf{C_B}$ does no further modification on this sanitized ciphertext, then $\mathsf{C_B}$ would retrieve K'_B from it. To maintain the functionality of the protocol Π', $\mathsf{F_A}$ must return Δ_B to $\mathsf{C_A}$ so that $\mathsf{C_A}$ can derive session key k_A with the same K'_B as $\mathsf{C_B}$. Similarly, $\mathsf{F_B}$ needs to rerandomize the ciphertext c_A, maul the key K_A to K'_A using the randomness Δ_A, and forward Δ_A back to $\mathsf{C_B}$. Clearly, both RFs are not transparent.

Furthermore, $\mathsf{F_B}$ is permitted only to rerandomize the incoming ciphertext c_B and should not maul the underlying K_B. If $\mathsf{F_B}$ were to alter K_B, it would be unable to return the randomness used in mauling K_B back to $\mathsf{C_A}$, causing both parties to disagree on K_B. Likewise, $\mathsf{F_A}$ should not change the underlying K_A of the incoming ciphertext c_A. However, rerandomizing the incoming ciphertexts alone is not sufficient to defend against input-trigger attacks. In the unauthenticated channel setting, an adversary could send an encryption of a triggering message to the parties. Rerandomizing this encryption does not sanitize its malicious payload, and the whole protocol is vulnerable to input-trigger attacks.

Our Treatment for Transparent RFs. To build an AKE protocol that is compatible with transparent RFs, which would also sanitize all incoming and outgoing messages for their cores, the protocol Π' needs an additional mechanism to ensure that both $\mathsf{C_A}$ and $\mathsf{C_B}$ would agree on the same K_A and K_B. As shown in Fig. 2, we ask $\mathsf{C_A}$ and $\mathsf{C_B}$ to negotiate an extra element R_AB by performing coin tossing, and this element allows $\mathsf{C_A}$ and $\mathsf{C_B}$ to reach the same session key. In specific, $\mathsf{C_A}$ first sends a commitment of a random R_A, then $\mathsf{C_B}$ returns a random R_B. Finally, $\mathsf{C_A}$ opens this commitment, and both parties get $R_\mathsf{AB} = R_\mathsf{A} R_\mathsf{B}$.

As the code for $\mathsf{F_A}$ and $\mathsf{F_B}$ is the same, we simplify the discussion by focusing solely on $\mathsf{C_A}$ having an RF $\mathsf{F_A}$, as detailed in Fig. 3. Both commitment schemes are instantiated identically for brevity. In specific, $\mathsf{F_A}$ first mauls the underlying K_B of the ciphertext c_B to $K'_\mathsf{B} = K_\mathsf{B} \Delta_\mathsf{B}$

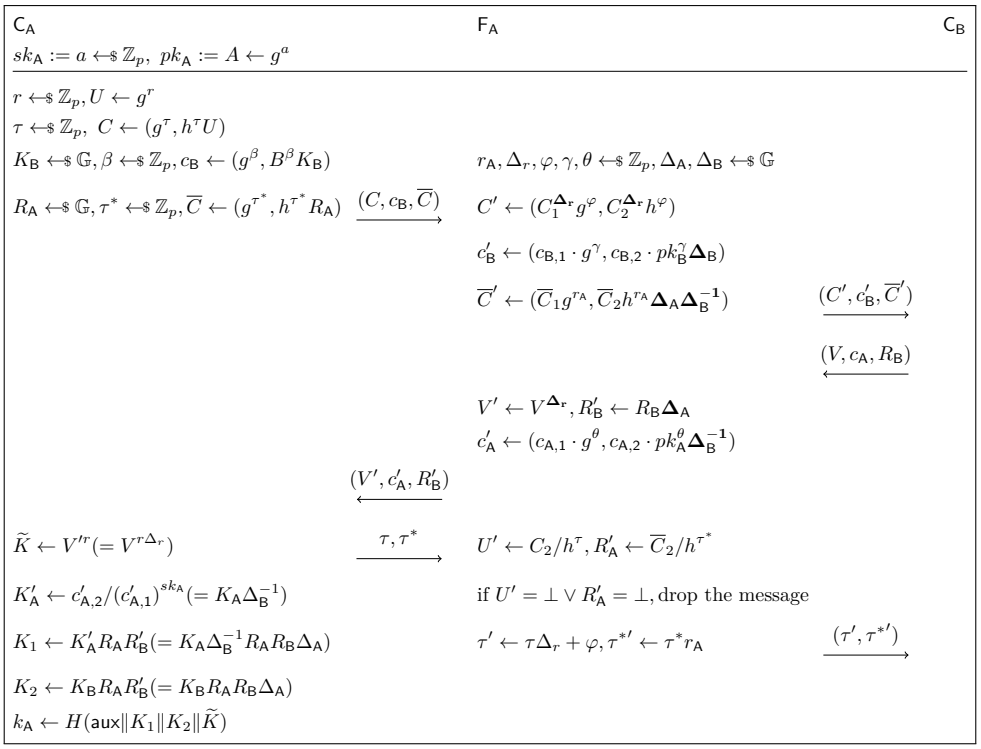


Figure 3: The operations of $\mathsf{F_A}$, where $h \in \mathbb{G}$ is a public parameter for both commitment schemes. Let $C = (C_1, C_2), \overline{C} = (\overline{C}_1, \overline{C}_2)$. For a pair of values $X_r = (E, F)$, let $X_{r,1} := E, X_{r,2} := F$.

and the underlying U and R_A of the commitments C and $\overline{C}$ to $U' = U^{\Delta_r}$ and $R'_\mathsf{A} = R_\mathsf{A} \Delta_\mathsf{A} \Delta_\mathsf{B}^{-1}$, respectively. It then mauls the underlying K_A of the ciphertext c_A to $K'_\mathsf{A} = K_\mathsf{A} \Delta_\mathsf{B}^{-1}$, and modifies V and R_B to $V' = V^{\Delta_r}$ and $R'_\mathsf{B} = R_\mathsf{B} \Delta_\mathsf{A}$. At last, $\mathsf{F_A}$ mauls the openings of commitments to maintain correctness. Now the session key derived by $\mathsf{C_A}$ is the same as the one by $\mathsf{C_B}$:

$$\begin{aligned}
k_\mathsf{A} &= H(\mathsf{aux} \| K'_\mathsf{A} R_\mathsf{A} R'_\mathsf{B} \| K_\mathsf{B} R_\mathsf{A} R'_\mathsf{B} \| V'^r) \\
&= H(\mathsf{aux} \| (K_\mathsf{A} \Delta_\mathsf{B}^{-1}) R_\mathsf{A} (R_\mathsf{B} \Delta_\mathsf{A}) \| K_\mathsf{B} R_\mathsf{A} (R_\mathsf{B} \Delta_\mathsf{A}) \| (V^{\Delta_r})^r) \\
&= H(\mathsf{aux} \| K_\mathsf{A} R_\mathsf{A} R_\mathsf{B} \Delta_\mathsf{A} \Delta_\mathsf{B}^{-1} \| K_\mathsf{B} R_\mathsf{A} R_\mathsf{B} \Delta_\mathsf{A} \| (V^r)^{\Delta_r}) \\
&= H(\mathsf{aux} \| K_\mathsf{A} R_\mathsf{A} R_\mathsf{B} \Delta_\mathsf{A} \Delta_\mathsf{B}^{-1} \| K_\mathsf{B} R_\mathsf{A} R_\mathsf{B} \Delta_\mathsf{A} (\Delta_\mathsf{B} \Delta_\mathsf{B}^{-1}) \| (U^s)^{\Delta_r}) \\
&= H(\mathsf{aux} \| K_\mathsf{A} (R_\mathsf{A} \Delta_\mathsf{A} \Delta_\mathsf{B}^{-1}) R_\mathsf{B} \| (K_\mathsf{B} \Delta_\mathsf{B}) (R_\mathsf{A} \Delta_\mathsf{A} \Delta_\mathsf{B}^{-1}) R_\mathsf{B} \| (U^{\Delta_r})^s) \\
&= H(\mathsf{aux} \| K_\mathsf{A} R'_\mathsf{A} R_\mathsf{B} \| K'_\mathsf{B} R'_\mathsf{A} R_\mathsf{B} \| U'^s) = k_\mathsf{B},
\end{aligned}$$

where aux does not include the transcript of protocol Π'' anymore, as the views of $\mathsf{C_A}$ and $\mathsf{C_B}$ on the transcript are different. We emphasize that removing the transcript from session key derivation allows an adversary to perform man-in-the-middle (MITM) attacks, manipulate messages like the RFs, and thereby impersonate an honest party. For protocols that are compatible with transparent RFs, such an impersonation attack is somewhat unavoidable [4], and applies to existing models of RFs [32, 24]. Note that no sensitive information will be leaked during an impersonation attack provided that the impersonated party is honest.

Generic Framework. Note that V can be viewed as an encapsulation of key U^s under the public key U. We generalize the protocol Π'' into an AKE framework based on KEM as shown in Fig. 4. Since $\mathsf{C_B}$ in Π'' has to send the KEM ciphertext V before receiving the public key U, the KEM ciphertext V shall be independent from the public key for the encapsulation algorithm. We call such a KEM *blind* and give the formal definition in Section 3.1.

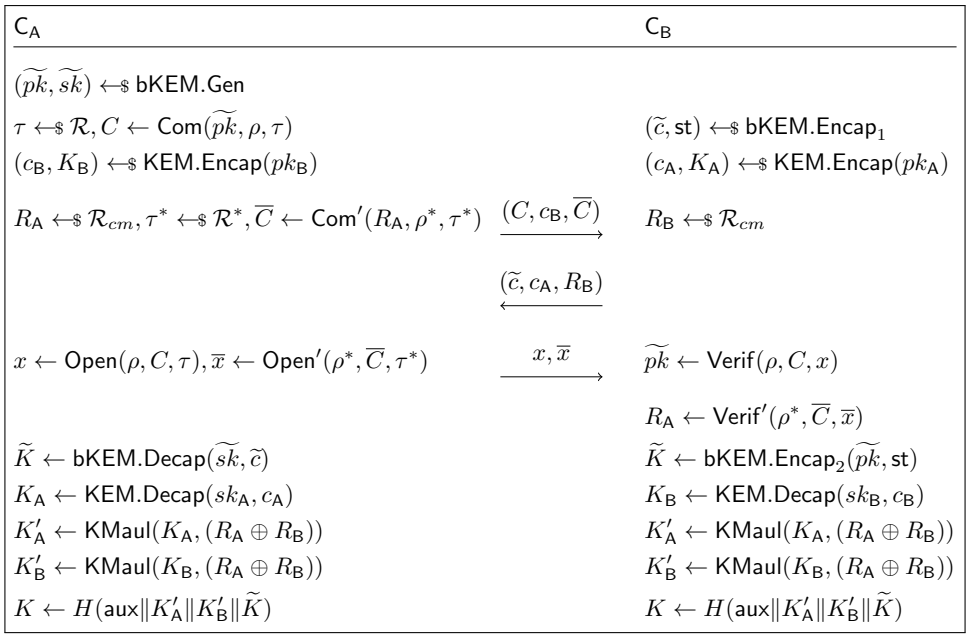


Figure 4: A general framework from Π''. In particular, bKEM is a blind KEM whose encapsulation algorithm Encap can be split into two independent parts. The first part Encap_1 generates the KEM ciphertext without public key, and the second part Encap_2 derives the key from public key $\widetilde{pk}$ and states in Encap_1. pk_A and pk_B denote the long-term public keys for $\mathsf{C_A}$ and $\mathsf{C_B}$, and $\mathsf{aux} = \mathsf{C_A} \| \mathsf{C_B} \| pk_\mathsf{A} \| pk_\mathsf{B}$.

In this generic framework of AKE protocol, the commitment schemes should be *malleable* and *rerandomizable* such that $\mathsf{F_A}$ could maul the underlying $\widetilde{pk}$ and R_A of the commitments C and $\overline{C}$ and rerandomize these commitments. Also, the blind KEM bKEM is required to be *key-malleable*, which means that its public key is malleable and a ciphertext under a mauled public key can be transformed into a ciphertext under the original public key. The regular KEM KEM is required to be *rerandomizable* and *ciphertext-malleable* (i.e., one can alter the encapsulated key by modifying the ciphertext). The algorithm KMaul is associated with the ciphertext malleability of KEM. To prove the protocol's security, we introduce a new security notion for blind KEM called *strong one-wayness*. Informally, given a blind KEM ciphertext c and a public key pk, it is hard to find (K, pk', c') such that $\mathsf{Decap}(sk, c') = \mathsf{Decap}(sk', c) = K$ where sk and sk' are the secret keys for pk and pk'. In particular, bKEM should satisfy this property.

In the same way that protocols by Chakraborty et al. [24, 23] rely on the trusted setup assumption, we assume the generation of public parameters and long-term key pairs in this protocol does not suffer from subversion attacks. In fact, removing or weakening this trusted setup assumption is a non-trivial task, and has been left as an open problem in [24].

2.3 Related Work

Here we give an overview of previous countermeasures against subversion attacks. RFs have been used to protect various cryptographic schemes including signatures [5], message transmission protocols [32], zero-knowledge proofs [37], and general MPC protocols [19, 20]. Existing AKE protocols with RFs [32, 15, 46] are proved in the standalone models. Specifically, Dodis et al. [32] proposed a mutually authenticated key agreement protocol that is compatible with

transparent RFs. Bossuat et al. [15] presented RFs with public-private key pairs that allow sanitizing the message of the handshake protocol without requiring underlying primitives to be rerandomizable. Recently, Liu et al. [46] proposed a generic subversion-resilient AKE construction with RFs under the classical game-based model, and Liu et al. [48] introduced a modified TLS 1.3 handshake protocol with RFs following Arnold et al.'s model [3]. However, these two protocols do not consider input-trigger attacks and their RFs are not transparent. Moreover, Liu et al. [47] proposed a PAKE protocol with transparent RFs that UC-realizes a relaxed PAKE functionality under Arnold et al.'s model [3].

The self-guarding mechanism by Fischlin and Mazaheri [35] assumes a secure initialization phase of protocols where parties are not subverted, serving as "security anchors". In comparison to RFs, it does not rely on an online third party. Moreover, a self-guarding mechanism is designed specifically to protect a key exchange protocol based on a physically unclonable function. Thus it could be a potential alternative to thwart subversions against AKE protocols.

Russell et al. [51] proposed the *watchdog* model to effectively detect the subversion attacks. This involves a detector "watchdog" that tests whether implementations of a cryptographic primitive strictly comply with its specification. Given the specification of a cryptographic scheme and its real-world implementation, a watchdog is allowed to detect whether the implementation is subverted. This method needs no trusted source of randomness (like the RFs) and is not limited by the samples collected during a clean initial phase (as in the self-guarding mechanism). A caveat is that it is challenging to resist the input-trigger attacks in this model. Over these years, watchdogs have been applied in many cryptographic primitives, such as one-way permutations [51], digital signatures [27, 10], authenticated encryption with associated data [11], pseudorandom generator [51], public-key encryption [52, 12], and random oracles [53].

3 Definitions and Components

Notations. Let g be a generator of the cyclic group $\mathbb{G}$ of prime order p. Let $y \leftarrow\!\!\$\ \mathcal{Y}$ denote sampling a uniformly random element y from $\mathcal{Y}$. $x := z$ means that the value of z is assigned to x. For a probabilistic algorithm $f(x)$ with input x, $y \leftarrow\!\!\$\ f(x)$ represents a random output of $f(x)$ and $y \leftarrow f(x;r)$ represents the output of $f(x)$ using randomness r. For a positive integer N, $[N]$ denotes the integer set $\mathcal{X} := \{1, 2, 3, \cdots, N\}$. Let λ denote the security parameter, $\mathsf{negl}(\lambda)$ denote a negligible function of λ and $p(\lambda)$ denote a polynomial of λ. For all polynomials $p(\lambda)$ and all sufficiently large λ, it holds that $\mathsf{negl}(\lambda) < \frac{1}{p(\lambda)}$.

3.1 Key Encapsulation Mechanism

A key encapsulation mechanism (KEM) [55] consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda)$ takes as input security parameter λ and produces public parameters pp, including the key space $\mathcal{K}$, public key space $\mathcal{PK}$, private key space $\mathcal{SK}$ and ciphertext space $\mathcal{C}$.
- $\mathsf{Gen}(\mathsf{pp})$ takes as input a public parameter pp and outputs a key pair (pk, sk).
- $\mathsf{Encap}(pk)$ takes as input a public key pk and outputs a ciphertext c and a key K.
- $\mathsf{Decap}(sk, c)$ takes as input a private key sk and a ciphertext c and outputs a key K.

Correctness. For all $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$, $(pk, sk) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{pp})$ and $(c, K) \leftarrow\!\!\$\ \mathsf{Encap}(pk)$, it holds that $\Pr[\mathsf{Decap}(sk, c) \neq K] \leq \mathsf{negl}(\lambda)$.

We define blindness for KEM as follows. In fact, the universal decryptability for KEM with no tags in [25] implies this property.

Definition 1 (Blindness). *For a KEM* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$*, we say* KEM *is blind if the* Encap *algorithm consists of following sub-algorithms.*

- *The randomness generation algorithm* $\mathsf{Rg}(\mathsf{pp})$*: takes as input a public parameter* pp*, outputs a randomness* r*.*
- *The key generation algorithm* $\mathsf{Kg}(pk, r)$*: takes as input a public key* pk *and a randomness* r*, outputs a key* K*.*
- *The ciphertext generation algorithm* $\mathsf{Cg}(r)$*: takes as input a randomness* r*, outputs a ciphertext* c*.*

and for all $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$*,* $r \leftarrow\!\!\$\ \mathsf{Rg}(\mathsf{pp})$*,* $c \leftarrow \mathsf{Cg}(r)$ *and* $(pk, sk) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{pp})$*, we have* $\Pr[\mathsf{Decap}(sk, c) \neq \mathsf{Kg}(pk, r)] \leq \mathsf{negl}(\lambda)$*.*

The malleability of the encapsulated key and the public key in a ciphertext is defined as follows. In particular, the definition of key malleability is adapted from the definition by Dodis et al. [32] for public-key encryption.

Definition 2 (Ciphertext malleability)**.** *For a KEM* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$*, we say* KEM *is ciphertext-malleable if following conditions hold.*

- *There exists a pair of algorithms* $(\mathsf{CMaul}, \mathsf{KMaul})$ *as below.*
  - $\mathsf{CMaul}(c, r)$*: takes as input a ciphertext* c *and a randomness* $r \in \mathcal{R}_{cm}$*, where* $\mathcal{R}_{cm}$ *is the randomness space, and outputs a new ciphertext* c'*;*
  - $\mathsf{KMaul}(K, r)$*: takes as input a key* K *and a randomness* $r \in \mathcal{R}_{cm}$*, and outputs a new key* K'*;*
- *For* $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$*,* $(pk, sk) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{pp})$*,* $(c, K) \leftarrow\!\!\$\ \mathsf{Encap}(pk)$ *and any* $r \in \mathcal{R}_{cm}$*, we have* $\Pr[\mathsf{Decap}(sk, \mathsf{CMaul}(c, r)) \neq \mathsf{KMaul}(K, r)] \leq \mathsf{negl}(\lambda)$*;*
- *Let* $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$*,* $(pk, sk) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{pp})$ *and* $K_0 \in \mathcal{K}$*. For* $(c, K) \leftarrow\!\!\$\ \mathsf{Encap}(pk)$*,* $r \leftarrow\!\!\$\ \mathcal{R}_{cm}$*,* $K' \leftarrow \mathsf{KMaul}(K_0, r)$*, the distributions of* K *and* K' *are identical.*

Definition 3 (Key malleability)**.** *For a KEM* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$*, we say* KEM *is key-malleable if following conditions hold.*

- *The key pair* (pk, sk) *output by* Gen *is uniformly distributed over the space of valid key pairs.*
- *For a public key* pk*, there is a unique corresponding private key* sk*;*
- *There exists a pair of algorithms* $(\mathsf{CKeyMaul}, \mathsf{KeyMaul})$ *as below.*
  - $\mathsf{CKeyMaul}(c, r)$*: takes as input a ciphertext* c *and a randomness* $r \in \mathcal{R}_{km}$*, and outputs a new ciphertext* c'*;*
  - $\mathsf{KeyMaul}(pk, r)$*: takes as input a public key* pk *and a randomness* $r \in \mathcal{R}_{km}$*, where* $\mathcal{R}_{km}$ *is the randomness space, outputs a new public key* pk'*;*
- *Let* (pk, sk) *and* (pk', sk') *be two public/private key pairs such that* $pk' = \mathsf{KeyMaul}(pk, r)$ *for* $r \in \mathcal{R}_{km}$*. For* $(c, K) \leftarrow\!\!\$\ \mathsf{Encap}(pk')$*, we have* $\Pr[\mathsf{Decap}(sk, \mathsf{CKeyMaul}(c, r)) \neq \mathsf{Decap}(sk', c)] \leq \mathsf{negl}(\lambda)$*;*
- *For any* $pk \in \mathcal{PK}$*,* $r \leftarrow\!\!\$\ \mathcal{R}_{km}$*, we have* $pk' \leftarrow \mathsf{KeyMaul}(pk, r)$ *is uniformly distributed over the public key space* $\mathcal{PK}$*.*

Also, we adapt the definition by Faonio et al. [34] for public-key encryption, and define perfect rerandomizability for KEM as follows.

Definition 4 (Perfect rerandomizability)**.** *For a KEM* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ *is perfectly rerandomizable if for any* $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$ *and* $(pk, sk) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{pp})$*, the following conditions hold:*

- *There exists an algorithm* $\mathsf{Rer}(c, r)$ *that takes as input a ciphertext* c *and a randomness* $r \in \mathcal{R}_{re}$*, where* $\mathcal{R}_{re}$ *is the randomness space, and outputs a new ciphertext* c'*;*

$\mathbf{Exp}$ OW-PCA$(\mathcal{A}, \lambda, \mathsf{KEM})$	$\mathsf{PCO}_{\mathsf{KEM},sk}(c, K)$
1 : $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$	1 : $K^* \leftarrow \mathsf{Decap}(sk, c)$
2 : $(pk, sk) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{pp})$	2 : **return** $K == K^*$
3 : $(c, K) \leftarrow\!\!\$\ \mathsf{Encap}(pk)$	
4 : $K' \leftarrow \mathcal{A}^{\mathsf{PCO}_{\mathsf{KEM},sk}(\cdot,\cdot)}(pk, c)$	
5 : **return** $K == K'$	

Figure 5: The OW-PCA game for $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$.

- *Given a ciphertext c s.t. $(c, \cdot) \in \mathsf{Encap}(pk)$, for any $r \leftarrow\!\!\$\ \mathcal{R}_{re}$, $c' \leftarrow \mathsf{Rer}(c, r)$ and $(c_0, \cdot) \leftarrow\!\!\$\ \mathsf{Encap}(pk)$, the distributions of c_0 and c' are identical.*
- *For any $r \leftarrow\!\!\$\ \mathcal{R}_{re}$, ciphertexts c and $c' \leftarrow \mathsf{Rer}(c, r)$, it holds that $\mathsf{Decap}(sk, c') = \mathsf{Decap}(sk, c)$.*
- *For any (computationally unbounded) adversary $\mathcal{A}$,*

$$\Pr[(c, \cdot) \notin \mathsf{Encap}(pk) \wedge \mathsf{Decap}(sk, c) \neq \perp | c \leftarrow \mathcal{A}(pk)] \leq \mathsf{negl}(\lambda).$$

The first condition, combined with the last one, indicates that if a (potentially maliciously generated) ciphertext decrypts correctly, the distribution of a rerandomized ciphertext is statistically close to that of a fresh ciphertext [34].

We recall a practical security notion of KEM, called *one-wayness under plaintext checking attacks* (OW-PCA) [56] as below.

Definition 5 (One-wayness under plaintext checking attacks [56])**.** *For a KEM $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$, we say KEM satisfies one-wayness under plaintext checking attacks (OW-PCA) if for any PPT adversary $\mathcal{A}$, $\mathrm{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{KEM}}(\mathcal{A}, \lambda) := \Pr[\mathsf{OW\text{-}PCA}(\mathcal{A}, \lambda, \mathsf{KEM}) = 1] \leq \mathsf{negl}(\lambda)$, where game $\mathsf{OW\text{-}PCA}(\mathcal{A}, \lambda, \mathsf{KEM})$ is defined in Fig. 5.*

We define a new notion called strong one-wayness for blind KEM as follows.

Definition 6 (Strong one-wayness)**.** *For a blind KEM $\mathsf{bKEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ where $\mathsf{Encap} = (\mathsf{Rg}, \mathsf{Kg}, \mathsf{Cg})$, let $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda), (pk, sk) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{pp}), r \leftarrow\!\!\$\ \mathsf{Rg}(\mathsf{pp})$ and $c \leftarrow \mathsf{Cg}(r)$, we say bKEM satisfies strong one-wayness if for any PPT adversary $\mathcal{A}$, it holds that $\mathrm{Adv}^{\mathsf{stOW}}_{\mathsf{bKEM}}(\mathcal{A}, \lambda) := \Pr[K = \mathsf{Kg}(pk', r) = \mathsf{Decap}(sk, c') | (K, pk', c') \leftarrow \mathcal{A}(pk, c)] \leq \mathsf{negl}(\lambda)$.*

3.2 Commitment Scheme

A commitment scheme consists of the following algorithms:

- $\mathsf{KGen}(1^\lambda)$ takes as input security parameter λ, and outputs the public parameter ρ, which includes the message space $\mathcal{M}_{com}$, randomness space $\mathcal{R}_{com}$, commitment space $\mathcal{C}_{com}$ and possibly the common reference string (CRS).
- $\mathsf{Com}(m, \rho, r)$ takes as input a message $m \in \mathcal{M}_{com}$, the public parameter ρ and randomness $r \in \mathcal{R}_{com}$, and outputs a commitment $C \in \mathcal{C}_{com}$.
- $\mathsf{Open}(\rho, C, r)$ takes as input the public parameter ρ, commitment C and randomness r, and outputs an opening x.
- $\mathsf{Verif}(\rho, C, x)$ takes as input the public parameter ρ, a commitment C and opening x, and outputs a message m or $\perp$.

Correctness. For all $\rho \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda), m \in \mathcal{M}_{com}, r \leftarrow\!\!\$\ \mathcal{R}_{com}, C \leftarrow \mathsf{Com}(m, \rho, r)$ and $x \leftarrow \mathsf{Open}(\rho, C, r)$, it holds that $\Pr[\mathsf{Verif}(\rho, C, x) \neq m] \leq \mathsf{negl}(\lambda)$.

As a basic security requirement, a commitment scheme should be binding and hiding. The formal definitions of these two properties are given in Appendix A.3.

We recall the definition of tightness for commitment as below.

Definition 7 (Tightness [32]). *A commitment scheme* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$ *is tight if for any* $\rho \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$*, commitment* C *and message* m*, there exists only one opening* x *such that* $\mathsf{Verif}(\rho, C, x) = m$*.*

We define the malleability and rerandomizability for a commitment scheme as follows. In particular, the definition of rerandomizability follows that in [32], while the definition of malleability does not require the plaintext space to be a group of order p, extending that in [32].

Definition 8 (Malleability). *Let* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$ *be a commitment scheme. Let* $\mathcal{R}_{mcom}$ *be the randomness space for mauling. We say* Π_{Com} *is malleable if there exist three efficient algorithms* $(\mathsf{MaulCom}, \mathsf{mMaul}, \mathsf{xMaul})$ *such that for* $\rho \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$*, any commitment* $C \in \mathcal{C}_{com}$ *and its opening* x*, let* $r \leftarrow\!\!\$\ \mathcal{R}_{mcom}$*,* $m' \leftarrow \mathsf{mMaul}(\mathsf{Verif}(\rho, C, x), r)$*,* $C' = \mathsf{MaulCom}(C, r)$ *and* $x' \leftarrow \mathsf{xMaul}(x, r)$*, it holds that* $m' = \mathsf{Verif}(\rho, C', x')$*. Moreover,* m' *is uniformly distributed over* $\mathcal{M}_{com}$*.*

Definition 9 (Rerandomizability [32]). *A commitment scheme* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$ *is rerandomizable if there exist two efficient algorithms* $(\mathsf{Rerand}, \mathsf{OpenRerand})$*, such that*

- *for any commitment* $C \in \mathcal{C}_{com}$ *and* $r \leftarrow\!\!\$\ \mathcal{R}_{com}$*,* $C' \leftarrow \mathsf{Rerand}(C, r)$ *is a uniformly random value in commitment space; and*
- *for* $\rho \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$*, any commitment* $C \in \mathcal{C}_{com}$ *and opening* x*, it holds that* $\mathsf{Verif}(\rho, C, x) = \mathsf{Verif}(\rho, C', x')$ *and* $x' \leftarrow \mathsf{OpenRerand}(x, r)$*.*

To prove the UC security of the proposed AKE, we require the underlying commitments to be extractable, which implies the perfect binding property [24].

Definition 10 (Extractability). *A commitment scheme* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$ *is extractable if following conditions hold:*

- *The* KGen *algorithm outputs both the public parameter* ρ *and an extraction key* ek*, and there exists an efficient algorithm* $\mathsf{Extract}$ *that takes as input a commitment* C *and an extraction key* ek *and outputs a message* m*;*
- *For* $(\rho, ek) \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$*, any* $m \in \mathcal{M}_{com}$*,* $r \in \mathcal{R}_{com}$*,* $C \leftarrow \mathsf{Com}(m, \rho, r)$ *and* $x \leftarrow \mathsf{Open}(\rho, C, r)$*,* $\Pr[\mathsf{Verif}(\rho, C, x) = m \wedge \mathsf{Extract}(ek, C) \neq m] \leq \mathsf{negl}(\lambda)$*.*

4 The srUC Model

In this section, we present the srUC model in the setting of unauthenticated communication. In particular, we follow Canetti and Krawczyk's approach [18] that casts the *unauthenticated-links model* (UM) by Bellare et al. [7] in the UC model. That is, the UM is cast as a "hybrid model" with restricted access to the ideal authenticated communication functionality $\mathcal{F}_{\text{AUTH}}$[1] [16]. For each ordered pair of parties, access to $\mathcal{F}_{\text{AUTH}}$ is restricted to a single use throughout the computation.

[1] $\mathcal{F}_{\text{AUTH}}$ guarantees authenticity of the communication content. Informally, $\mathcal{F}_{\text{AUTH}}$ operates by receiving a message $(\text{SEND}, sid, \mathcal{P}', m)$ from party $\mathcal{P}$, sending $(\text{SENT}, sid, \mathcal{P}, \mathcal{P}', m)$ to both $\mathcal{P}'$ and the adversary. Note that the adversary may delay the message sent to $\mathcal{P}'$, or determine its content and receiver identity (if undelivered) via a corruption request $(\text{CORRUPT}, sid, m', \mathcal{P}'')$.

Table 2: Corruption patterns in the real world and their translation to those of a party in the ideal world [24].

Core \ RF	Honest	SemiHonest	Malicious
Honest	Honest	Honest	Isolate
Specious	Honest	Malicious	Malicious
Malicious	Malicious	Malicious	Malicious

In other words, each party can only send one ideally authenticated message to every other party. Typically, this message includes initial information for establishing a long-term authentication mechanism, which corresponds to the "initialization function" [7, 18] that is executed before the initiation of a protocol and securely generates the long-term information for parties. During subsequent executions, the adversary takes full control of the communication channel and can freely modify messages transmitted between RFs. We stress that all protocols presented in this work are described within the $\mathtt{UM}$. Moreover, given that an RF is typically installed on the same machine as the core, it is reasonable to assume that communication between a core and its RF is secure.

Remark. Since all the protocols in [24] are depicted in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model, where $\mathcal{F}_{\mathsf{SAT}}$ denotes the ideal functionality for sanitizable authenticated communication, the definitions of the properties for RFs in the srUC model and the associated lemma (i.e., Lemma 1 in [24]) are also described in the setting of $\mathcal{F}_{\mathsf{SAT}}$-hybrid model. For AKE protocols, which run over unauthenticated channels, we need to tweak these definitions and the lemma in the srUC model.

4.1 Corruption Cases

In the srUC model, each party consists of a core and an RF. The core holds the secret inputs of the party and generates the messages of the protocol, while the RF does not have access to the secrets of the core and sanitizes the incoming/outgoing messages for the core with randomness. A core could be honest, specious or malicious, and an RF can be honest, semi-honest or malicious. The so-called specious corruption on a core only captures a specific type of subversion attacks that would not be detected via black-box test (see Appendix B for the formal description of specious corruption), as it is unlikely to achieve any security in the presence of arbitrary subversion attacks for certain ideal functionalities [24].

The combination of corruptions on core and RF results in 9 corruption cases which give rise to an honest, isolate or malicious party, as shown in Table 2. Each cell represents a corruption state of the party under the corresponding corruption case of the core and RF. For instance, if a core is speciously corrupted and an RF is honest, then the RF would faithfully sanitize messages for the core and the whole party can be viewed as an honest one. As a special case, when a core is honest and an RF is malicious, the composed party is isolated. The standard behavior of an ideal functionality $\mathcal{F}$ upon an isolate corruption is to perform a malicious corruption of the party in $\mathcal{F}$. However, as mentioned by Chakraborty et al. [23], in the unauthenticated communication setting, an isolate corruption is treated as part of the active adversary and therefore can be safely removed from the security analysis. For corruption cases that result in the same corruption on a party, one only needs to consider the simulation of the case (highlighted in Table 2) that gives the adversary the most capabilities.

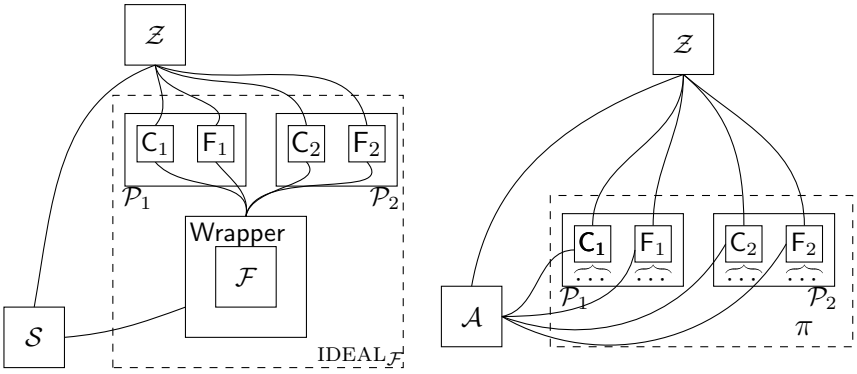


Figure 6: The execution of an ideal protocol IDEAL$_{\mathcal{F}}$ (left) and a protocol π (right) in the srUC model. The wrapper in IDEAL$_{\mathcal{F}}$ forwards the input of C_i to the interface for $\mathcal{P}_i$ on $\mathcal{F}$, and returns the output of $\mathcal{P}_i$ to C_i. Also, the wrapper is responsible for translating corruptions on C_i and F_i into a corruption on $\mathcal{P}_i$. The cores and RFs in protocol π may have subroutines denoted by "$\cdots$".

4.2 Wrapped Subversion-Resilient UC Security

In the srUC model, there are two types of ideal functionalities: sanitizable and regular ideal functionalities. The former provides explicit interfaces for the cores and RFs, while the latter are essentially ideal functionalities as in the UC model, interacting only with parties without being aware of cores and RFs.

To establish a structured connection between a regular ideal functionality $\mathcal{F}$ and cores/RFs, Chakraborty et al. [24] introduced the concept of a *wrapper* (denoted as Wrap) and *wrapped ideal functionality* Wrap($\mathcal{F}$). It runs the regular ideal functionality $\mathcal{F}$ internally, which is denoted by Wrap($\mathcal{F}$). As shown in Fig. 6, Wrap is used to provide the same interfaces for cores and RFs as in the protocol, forward the input of a core to the interface for the party on $\mathcal{F}$, and return the party's output to the core. Also, it translates corruptions of the core and RF into a corruption of the party following the corruption translation table (i.e., Table 2). The definition of UC security with subversion resilience for a regular ideal functionality is shown below, where specious environments/adversaries are formally defined in Appendix B. Roughly, in the srUC model, a specious environment would prepare a list of specious corruptions for the adversary, and a specious adversary would do specious corruptions in the list over the cores. Note that all specious corruptions in the srUC model are assumed to be static.

Definition 11 (Wrapped subversion-resilient UC security [24])**.** *Let $\mathcal{F}$ be an ideal functionality for n parties $\mathsf{P}_1, \cdots, \mathsf{P}_n$. Let Π be a sanitizing protocol with n cores $\mathsf{C}_1, \cdots, \mathsf{C}_n$ and n firewalls $\mathsf{F}_1, \cdots, \mathsf{F}_n$. Let $\mathcal{G}$ be a sanitizable ideal functionality that can be used by Π. We say that Π **wsrUC-realizes** $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if Π UC-realizes* Wrap($\mathcal{F}$) *in the $\mathcal{G}$-hybrid model with the restriction that we only consider specious environments and specious adversaries.*

4.3 Properties of Reverse Firewalls

An RF [49] is initially defined as a stateful entity that maintains a list of session identifiers and associated state information. In particular, the former is used to identify different sessions and the latter is required to sanitize the corresponding session correctly.

One central property of RFs is strong sanitation. It ensures that no environment capable of performing specious corruptions on an honest core in the real world can distinguish between a protocol execution from one where the honest core is replaced by an incorruptible core (which

remains honest during specious corruption), provided the honest core's firewall is also honest, i.e., the combination of a specious core and an honest RF is indistinguishable from the combination of an honest core and an honest RF. Let $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, r)$ represent the random variable that describes the output of executing a protocol Π with the environment $\mathcal{Z}$ and adversary $\mathcal{A}$, using λ as the security parameter and r as input. Let $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, r)\}_{r\in\{0,1\}^*}$ [16]. The formal definition of indistinguishable distribution ensembles is provided in Appendix A.2. The definition of strong sanitation below is the same as in [24] except that the protocol runs in the UM, rather than the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model.

Definition 12 (Strong sanitation)**.** *Let $\mathcal{P} = (\mathsf{C}, \mathsf{F})$ denote a party with a core C and an honest RF F. Let $\widehat{\mathsf{C}}$ denote the corresponding incorruptible core that would ignore any specious corruption. Let Π_i be a protocol running in the* UM *with the i-th party $\mathcal{P}_i = (\mathsf{C}, \mathsf{F})$, and all other parties are dummy. Let Π_i' be the protocol identical to Π_i, except for the i-th party $\mathcal{P}_i = (\widehat{\mathsf{C}}, \mathsf{F})$. F is* strongly sanitizing *if, for all PPT environments $\mathcal{Z}$ which are allowed to do specious corruptions of C, it holds that $\text{EXEC}_{\Pi_i,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\Pi_i',\mathcal{A},\mathcal{Z}}$, where $\mathcal{A}$ is the dummy adversary.*

Another important property of RFs is computational transparency. It captures the fact that an honest core without RF is indistinguishable from an honest core with RF. In [24], the definition of this property is limited to protocols in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model, as there is no complex interaction between $\mathcal{F}_{\mathsf{SAT}}$ and the RF, and it is feasible to replace an RF with an identity RF that only forwards messages for the core. Similar statements apply to the protocol in the UM, and we present the definition of computational transparency as follows.

Definition 13 (Computational transparency)**.** *Let ID be an "identity" RF which only forwards messages for the core. Let Π_i be a protocol running in the* UM *with the i-th party $\mathcal{P}_i = (\mathsf{C}, \mathsf{F})$, and all other parties are dummy. Let Π_i' be the same as Π_i except that the i-th party is $\mathcal{P}_i' = (\mathsf{C}, \mathsf{ID})$. We say that F is* computationally transparent *if, for all PPT environments $\mathcal{Z}$ that do not corrupt C or F/ID, we have $\text{EXEC}_{\Pi_i,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\Pi_i',\mathcal{A},\mathcal{Z}}$, where $\mathcal{A}$ is a dummy adversary.*

Using the above two properties of RFs, the corruption cases requiring simulation can be further reduced or weakened, as per Lemma 2 in [24]. We restate this lemma in the UM as follows, while treating isolate corruptions as MITM attacks. Its proof is provided in Appendix C, which is similar to that in [24].

Lemma 1. *Let Π be a protocol of n parties $(\mathsf{C}_i, \mathsf{F}_i)$ $(i = 1, 2, \cdots, n)$. Assume that Π is executed in the* UM*, with a secure channel between each core and its RF. If all the RFs in Π are strongly sanitizing, then it suffices to consider the following corruption cases: (1) Honest core & semi-honest RF. (2) Malicious core & malicious RF. Moreover, if all the RFs are computationally transparent, then it suffices to consider the following corruption cases: (1) Honest core & Honest RF. (2) Malicious core & malicious RF.*

5 Ideal Functionality for AKE

Our ideal functionality $\mathcal{F}_{\mathsf{AKE}}$ for AKE is adapted from $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ by Jarecki et al. [43], both illustrated in Fig. 7. We first give a brief overview of $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$, and then explain the rationales behind the modifications on this ideal functionality.

Overview of $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$. First, $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ makes some syntactic adjustments of the original key exchange functionality $\mathcal{F}_{\mathsf{KE}}$ [18] to the user-server setting. The key exchange functionality has three basic interfaces: UsrSession, SrvSession, and NewKey. Users initiate an AKE session via UsrSession, while servers use SrvSession. Both parties derive a session key by issuing a

Suppose that $\mathcal{P}, \mathcal{P}' \in \{\mathsf{U}, \mathsf{S}\}$. Let $\mathcal{S}$ denote the adversary.

Upon receiving a query (UsrSession, sid, $ssid$, S) from U:

- Send (UsrSession, sid, $ssid$, U, S) to $\mathcal{S}$.
- If $ssid$ was not used before by U, create session record $\langle ssid, \mathsf{U}, \mathsf{S}, \perp \rangle$ marked FRESH and send (UsrSession, sid, $ssid$, U, S) to $\mathcal{S}$.

Upon receiving a query (SrvSession, sid, $ssid$, U) from S:

- Send (SrvSession, sid, $ssid$, U, S) to $\mathcal{S}$.
- If $ssid$ was not used before by S, create session record $\langle ssid, \mathsf{S}, \mathsf{U}, \perp \rangle$ marked FRESH and send (SrvSession, sid, $ssid$, U, S) to $\mathcal{S}$.

Upon receiving a query (Compromise, sid, $\mathcal{P}$) from $\mathcal{S}$:

- Mark $\mathcal{P}$ COMPROMISED.
- If there is a record $\langle ssid, \mathcal{P}', \mathcal{P}, \kappa \rangle$ marked INTERFERED, then send $(ssid, \mathcal{P}', \mathcal{P}, \kappa)$ to $\mathcal{S}$.

Upon receiving a query (Impersonate, sid, $ssid$, $\mathcal{P}$) from $\mathcal{S}$: If there is a record $\langle ssid, \mathcal{P}', \mathcal{P}, \perp \rangle$ marked FRESH:

- If $\mathcal{P}$ is marked COMPROMISED, mark this record COMPROMISED.
- Else mark this record INTERFERED.

Upon receiving a query (NewKey, sid, $ssid$, $\mathcal{P}, \kappa^*$) from $\mathcal{S}$, where $|\kappa^*| = \lambda$, if there is a record $\mathsf{rec} = \langle ssid, \mathcal{P}, \mathcal{P}', \perp \rangle$, then:

- If $\mathcal{P}$ is corrupted and rec is not FRESH, or both $\mathcal{P}$ **and** $\mathcal{P}'$ are corrupted, set $\kappa \leftarrow \kappa^*$.
- If rec is COMPROMISED, or $\mathcal{P}$ or $\mathcal{P}'$ is corrupted, set $\kappa \leftarrow \kappa^*$.
- Else if rec is marked FRESH and there is a record $\langle ssid, \mathcal{P}', \mathcal{P}, \kappa' \rangle$ marked FRESH, set $\kappa \leftarrow \kappa'$.
- Else, pick $\kappa \leftarrow\!\!\$\ \mathcal{K}$, where $\mathcal{K}$ is the key space.

If rec is COMPROMISED, send κ to $\mathcal{S}$. Finally, update rec to $\langle ssid, \mathcal{P}, \mathcal{P}', \kappa \rangle$ and send $(sid, ssid, \kappa)$ to $\mathcal{P}$.

Figure 7: AKE functionality $\mathcal{F}_{\mathsf{AKE}}$, which includes all codes except the boxed text. $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ [43] includes all codes but the dark gray parts.

NewKey query. If both are honest, the functionality provides a uniform random key; otherwise, the adversary determines the key.

$\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ extends $\mathcal{F}_{\mathsf{KE}}$ by adding two adversarial behaviors: Compromise and Impersonate. Compromise models an adversary (referred to as $\mathcal{S}$) obtaining the private keys of a party. If $\mathcal{S}$ makes a (Compromise, sid, $\mathcal{P}$) query, then $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ marks $\mathcal{P}$ as COMPROMISED. $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$ handles *adaptive* compromise by which $\mathcal{S}$ is allowed to make a Compromise query at any time (before, during or after a session execution). Compared with corrupting a party $\mathcal{P}$, $\mathcal{S}$ is not allowed to control a COMPROMISED $\mathcal{P}$. However, $\mathcal{S}$ can impersonate $\mathcal{P}$ in sessions with another party $\mathcal{P}'$, which is captured by the Impersonate interface.

A query (Impersonate, sid, $ssid$, $\mathcal{P}$) from $\mathcal{S}$ models the impersonation of $\mathcal{P}$ (that has been labeled COMPROMISED) in session $\langle ssid, \cdot, \mathcal{P} \rangle$, and $\mathcal{S}$ could determine the session key. If party $\mathcal{P}$ is labeled COMPROMISED but not corrupted, $\mathcal{P}$'s sessions that are uncorrupted and not labeled COMPROMISED are not considered COMPROMISED. This captures KCI security, i.e., an adversary that compromises a party $\mathcal{P}$ can impersonate $\mathcal{P}$, but can not impersonate other parties to $\mathcal{P}$.

Changes in the AKE functionality. For clarity, we choose to add the session key κ into a record $\langle ssid, \mathcal{P}, \mathcal{P}', \kappa \rangle$ instead of using the COMPLETED label. If a session key is not determined yet, we set $\kappa = \perp$.

There are two major modifications to $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$. The first one involves the cases where the adversary determines the session key. In $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$, the adversary determines the session key of $\mathcal{P}$ in session $\langle ssid, \mathcal{P}, \mathcal{P}' \rangle$ if (1) the record $\langle ssid, \mathcal{P}, \mathcal{P}', \perp \rangle$ is marked COMPROMISED, or (2) $\mathcal{P}$ or $\mathcal{P}'$ is corrupted. By comparison, in $\mathcal{F}_{\mathsf{AKE}}$, the adversary is only allowed to control the session key of party $\mathcal{P}$ in session $\langle ssid, \mathcal{P}, \mathcal{P}' \rangle$ if (1) $\mathcal{P}$ is corrupted and the record $\langle ssid, \mathcal{P}, \mathcal{P}', \perp \rangle$ is not marked FRESH, or (2) both parties are corrupted. Specifically,

(i) The key of session $\langle ssid, \mathcal{P}, \mathcal{P}' \rangle$ is no longer determined by the adversary when (a) only $\mathcal{P}$ is corrupted and the record $\langle ssid, \mathcal{P}, \mathcal{P}', \perp \rangle$ is FRESH (i.e., $\mathcal{S}$ does not send a (IMPERSONATE, $sid, ssid, \mathcal{P}'$) query), or (b) only $\mathcal{P}'$ is corrupted, which captures security against key control. As shown by Dodis et al. [32], a corrupt implementation of either party has the potential to manipulate the value of session key without being detected. This change prevents any corrupted party interacting with an uncorrupted party from controlling the session key.

(ii) The adversary is only allowed to obtain, rather than determine, the session key of uncorrupted party $\mathcal{P}$ in a session $\langle ssid, \mathcal{P}, \mathcal{P}' \rangle$ whose record $\langle ssid, \mathcal{P}, \mathcal{P}', \cdot \rangle$ has been marked COMPROMISED. This change also captures security against key control: The COMPROMISED record $\langle ssid, \mathcal{P}, \mathcal{P}', \cdot \rangle$ implies that $\mathcal{P}'$ has been marked COMPROMISED and $\mathcal{S}$ has impersonated $\mathcal{P}'$ in session $\langle ssid, \mathcal{P}, \mathcal{P}' \rangle$. Therefore, $\mathcal{S}$ interacts with an uncorrupted party $\mathcal{P}$ in the name of $\mathcal{P}'$. The session key is unbiased due to security against key control. However, $\mathcal{S}$ is allowed to obtain the session key since it has impersonated $\mathcal{P}'$ by knowing the private key of $\mathcal{P}'$.

The second modification is related to weak forward secrecy. In $\mathcal{F}_{\mathsf{AKE\text{-}KCI}}$, the full forward secrecy is implicitly captured: the only behavior of $\mathcal{S}$ upon receiving a query (COMPROMISE, $sid, \mathcal{P}$) is to mark $\mathcal{P}$ as COMPROMISED, which means that once the session key κ has been generated, the COMPROMISE query does not help obtain κ. In $\mathcal{F}_{\mathsf{AKE}}$, weak forward secrecy is captured by strengthening the ability of $\mathcal{S}$. In specific, there is an additional check upon receiving a (COMPROMISE, $sid, \mathcal{P}$) query: If there is a record $\langle ssid, \mathcal{P}', \mathcal{P}, \kappa \rangle$ not marked FRESH, then sends κ to $\mathcal{S}$. The rationale for this design is as below.

In general, full forward secrecy ensures the security of a session regardless of COMPROMISE of either party after the session is completed. For an AKE protocol with implicit authentication and weak forward secrecy, a standard way to achieve full forward secrecy is to add key confirmation messages (derived from the session key) [38]. However, only the parties who possess the session key can compute the key confirmation message correctly, which makes it difficult for an RF to check and sanitize the key confirmation messages. For better compatibility of AKE protocols with RFs, we aim to achieve weak forward secrecy.

Weak forward secrecy ensures the security of a passively observed session regardless of COMPROMISE of either party. If a session $\langle ssid, \mathcal{P}, \mathcal{P}' \rangle$ was actively interfered in, and $\mathcal{P}'$ is COMPROMISED, the security of session keys is not guaranteed. In $\mathcal{F}_{\mathsf{AKE}}$, the COMPROMISED and INTERFERED marks of a record model the ability of $\mathcal{S}$ to actively interfere in sessions, where a record $\langle ssid, \mathcal{P}', \mathcal{P}, \cdot \rangle$ is marked INTERFERED if $\mathcal{S}$ makes a (IMPERSONATE, $sid, ssid, \mathcal{P}$) query but $\mathcal{P}$ is not marked COMPROMISED. A session that is not marked COMPROMISED or INTERFERED is always considered to be FRESH. A record $\langle ssid, \mathcal{P}', \mathcal{P}, \kappa \rangle$ being not FRESH implies that it was actively interfered in before κ is generated. In such a case, COMPROMISE of $\mathcal{P}$ would reveal κ by weak forward secrecy. Note that $\mathcal{F}_{\mathsf{khAKE}}$ in [39] also captures weak forward secrecy through the Compromise and Interfere actions of the adversary.

6 Generic Construction

6.1 Building Blocks

Our generic subversion-resilient AKE protocol srAKE, is built on a blind KEM bKEM, a regular KEM KEM and two commitment schemes $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$, $\Pi'_{\mathsf{Com}} = (\mathsf{KGen}', \mathsf{Com}', \mathsf{Open}', \mathsf{Verif}')$. KEM satisfies OW-PCA, bKEM satisfies strong one-wayness, and both Π_{Com} and Π'_{Com} are computationally hiding, tight and extractable. To support the RFs, we require that bKEM is key-malleable, KEM is ciphertext-malleable, both Π_{Com} and Π'_{Com} are rerandomizable. Moreover, Π_{Com} is malleable with respect to bKEM, and Π'_{Com} is malleable with respect to KEM. In particular, let $\mathcal{PK}_{\mathsf{bKEM}}$ be the public key space of bKEM, $\mathcal{R}_{cm}$ the randomness space of KEM, and $\mathcal{M}_{com}$, $\mathcal{M}'_{com}$ the message spaces of Π_{Com} and Π'_{Com}, respectively. Then, $\mathcal{PK}_{\mathsf{bKEM}} \subseteq \mathcal{M}_{com}$ and $\mathcal{R}_{cm} \subseteq \mathcal{M}'_{com}$ are required. Below are the formal definitions.

Definition 14 (Malleability with respect to a key-malleable KEM)**.** *Let* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$ *be a commitment scheme and* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ *be a key-malleable KEM with algorithms* $(\mathsf{CKeyMaul}, \mathsf{KeyMaul})$*. We say* Π_{Com} *is malleable with respect to* KEM *if (1)* Π_{Com} *is malleable, i.e., there exist three algorithms* $(\mathsf{MaulCom}, \mathsf{mMaul}, \mathsf{xMaul})$*; and (2) for any* pk *generated by* Gen *and* $r \in \mathcal{R}_{mcom} \cap \mathcal{R}_{km}$*, where* $\mathcal{R}_{mcom}$ *is the randomness space for mauling in* Π_{Com} *and* $\mathcal{R}_{km}$ *is the randomness space of* KEM*, it holds that* $pk \in \mathcal{M}_{com}$ *and* $\mathsf{mMaul}(pk, r) = \mathsf{KeyMaul}(pk, r)$*.*

For $X \in \{\mathsf{CMaul}, \mathsf{KMaul}, \mathsf{CKeyMaul}, \mathsf{KeyMaul}, \mathsf{MaulCom}, \mathsf{MaulCom}'\}$, algorithm X takes as input an element and a randomness, and outputs a new element. Let $\mathcal{M}_{in}$ and $\mathcal{R}_m$ be the element space and randomness space respectively, and $X(c, r_1 \oplus r_2)$ denote $X(X(c, r_1), r_2)$. For all $c \in \mathcal{M}_{in}$, $r_1, r_2 \in \mathcal{R}_m$, it is required that $X(c, r_1 \oplus r_2) = X(c, r_2 \oplus r_1)$. For all $r \in \mathcal{R}_m$, we say r^{-1} is the *inverse element* of r if $X(c, (r \oplus r^{-1})) = c$.

Definition 15 (Malleability with respect to a ciphertext-malleable KEM)**.** *Let* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$ *be a commitment scheme and* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ *be a ciphertext-malleable KEM with algorithms* $(\mathsf{CMaul}, \mathsf{KMaul})$*. We say* Π_{Com} *is malleable with respect to* KEM *if (1)* Π_{Com} *is malleable, i.e., there exist three algorithms* $(\mathsf{MaulCom}, \mathsf{mMaul}, \mathsf{xMaul})$*; and (2) it holds that* $\mathcal{R}_{cm} \subseteq \mathcal{M}_{com}$ *and* $\mathcal{R}_{mcom} = \mathcal{R}_{cm}$*. Moreover, for* $r, r_{cm} \in \mathcal{R}_{cm}$ *and any* K *generated by* Encap*, it holds that* $\mathsf{KMaul}(K, \mathsf{mMaul}(r_{cm}, r)) = \mathsf{KMaul}(K, r_{cm} \oplus r)$*.*

6.2 Protocol srAKE

Assume that srAKE is executed between two parties $\mathsf{U} = (\mathsf{C}_i, \mathsf{F}_i)$ and $\mathsf{S} = (\mathsf{C}_j, \mathsf{F}_j)$. The behaviors of U and S are respectively illustrated in Fig. 8 and Fig. 9, where $H : \{0,1\}^* \to \{0,1\}^w$ is a hash function with $w = p(\lambda)$. Note that the description of F_j is identical to that of F_i. The public parameters $\mathsf{pp} \leftarrow \mathsf{bKEM.Setup}(1^\lambda)$, $\mathsf{pp}' \leftarrow \mathsf{KEM.Setup}(1^\lambda)$, $\rho \leftarrow\!\$\ \mathsf{KGen}(1^\lambda), \rho' \leftarrow\!\$\ \mathsf{KGen}'(1^\lambda)$ are pre-shared by parties. At the beginning of the protocol, the cores C_i and C_j generate their long-term public/private key pairs (pk_i, sk_i) and (pk_j, sk_j) via $\mathsf{KEM.Gen}(\mathsf{pp}')$, then send the public keys to each other using $\mathcal{F}_{\text{AUTH}}$. This initialization phase is omitted in Fig. 8 and Fig. 9. To adhere to the UC notion of AKE modeled by $\mathcal{F}_{\mathsf{AKE}}$, we incorporate the session identifier *sid* and sub-session identifier *ssid* into the session key derivation process, a practice commonly employed in UC-secure AKE protocols [43, 39][2]. A textual representation of srAKE and the correctness analysis are shown in Appendix D.1 and Appendix D.2, respectively.

[2]As Gu et al. [40] mentioned, in UC conventions, the top-level state of a specific cryptographic application, such as long-term keys of a public-key encryption, is called its "session", whereas the specific instances that utilize these long-term keys are termed "sub-sessions".

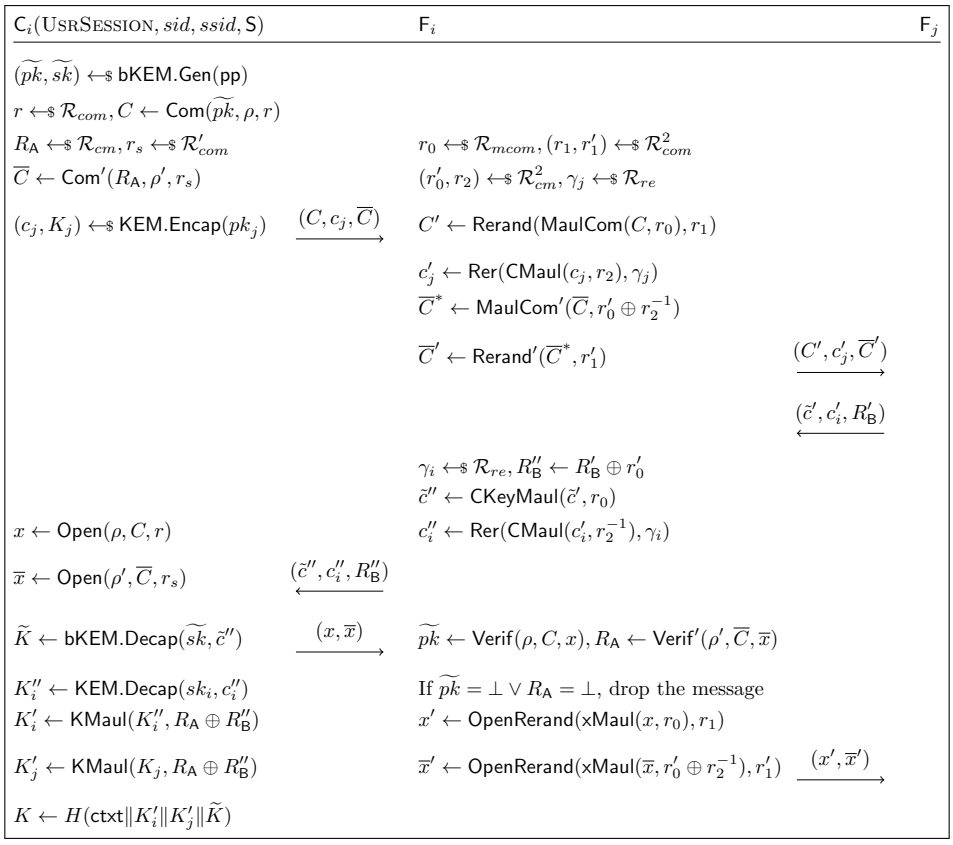


Figure 8: C_i and F_i in srAKE. Let $\mathsf{ctxt} = sid \| ssid \| \mathsf{aux}$ where $\mathsf{aux} := \mathsf{C}_i \| \mathsf{C}_j \| pk_i \| pk_j$.

Theorem 1. *Let* KEM *be a KEM,* bKEM *be a blind KEM,* Π_{Com} *and* Π'_{Com} *be two commitment schemes, all as defined above. The* srAKE *protocol wsrUC-realizes* $\mathcal{F}_{\mathsf{AKE}}$ *under static corruptions when* H *is modeled as an RO.*

Proof. By the definition of wsrUC security, we prove that srAKE UC-realizes $\mathsf{Wrap}(\mathcal{F}_{\mathsf{AKE}})$. In particular, we analyze various corruption cases for both parties. Since our protocol operates over unauthenticated channels, the adversary has control over the network channel and can insert, alter, or discard messages.

To simplify the corruption cases, we prove the strong sanitation (Definition 12) and computational transparency (Definition 13) of the RFs in srAKE. Since the isolate corruption collapses to MITM attacks in the setting of unauthenticated communication [22], it is not considered explicitly in our security analysis. Subsequently, by Lemma 1, we consider two corruption types: *both the core and firewall are honest*, resulting in an honest party; *or both are malicious*, resulting in a malicious party. These lead to two corruption cases for analysis in srAKE: one malicious and one honest party; or both parties being honest.

Lemma 2. *The firewalls* F_i *and* F_j *are strongly sanitizing.*

Proof. To prove the strong sanitation of F_i, we claim that for all specious environments $\mathcal{Z}$ which are not allowed to corrupt F_i, but are permitted to speciously corrupt C_i, $\textsc{Exec}_{\mathsf{srAKE},\mathcal{A},\mathcal{Z}} \approx \textsc{Exec}_{\mathsf{srAKE}^*,\mathcal{A},\mathcal{Z}}$ where srAKE and srAKE* are executed by dummy parties except for U which is either $(\mathsf{C}_i, \mathsf{F}_i)$ or $(\widehat{\mathsf{C}}_i, \mathsf{F}_i)$ where $\widehat{\mathsf{C}}_i$ is an incorruptible core. If $\mathcal{Z}$ sends a specious corruption to C_i, then srAKE runs with a specious core $\widetilde{\mathsf{C}}_i$ while srAKE* still runs with $\widehat{\mathsf{C}}_i$. Following

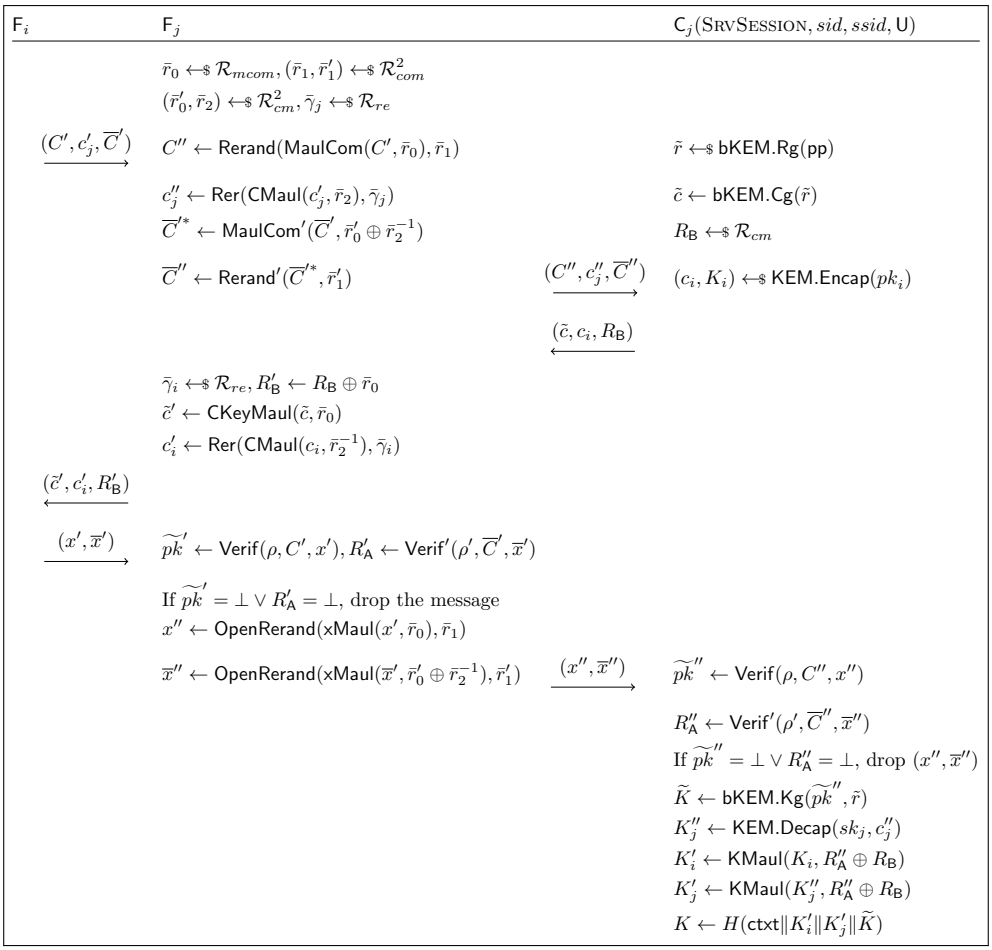


Figure 9: F_j and C_j in srAKE.

Chakraborty et al. [22], we first demonstrate that an adversary cannot mount input-triggered attacks against C_i. We then prove that C_i reveals no information, i.e., the messages output by F_i remain unbiased.

For the incoming message $v'' := (\tilde{c}'', c''_i, R''_{\mathsf{B}})$ of C_i: R''_{B} is uniformly random due to the use of r'_0. By the key malleability of bKEM, it holds that $\widetilde{K} = \mathsf{bKEM.Decap}(\widetilde{sk}, \tilde{c}'') = \mathsf{bKEM.Decap}(\widetilde{sk}', \tilde{c}')$ where $\tilde{c}'' = \mathsf{CKeyMaul}(\tilde{c}', r_0)$ and $\widetilde{sk}'$ is the corresponding private key of $\widetilde{pk}'$. Since $\widetilde{pk}' = \mathsf{KeyMaul}(\widetilde{pk}, r_0)$ is uniformly random, we have that $\widetilde{K} = \mathsf{bKEM.Decap}(\widetilde{sk}', \tilde{c}')$ computed by C_i is uniformly random for a given $\tilde{c}'$. Therefore, $\tilde{c}''$ received by C_i is also uniformly random (for a given $\widetilde{sk}$). Let $(\hat{c}_i, \widehat{K}_i) \leftarrow\$ \mathsf{Encap}(pk_i)$. The distribution of $c''_i = \mathsf{Rer}(\mathsf{CMaul}(c'_i, r_2^{-1}), \gamma_i)$ is the same as that of $\hat{c}_i$ by the perfect rerandomizability of KEM. Moreover, the distributions of the underlying key K''_i of c''_i and $\widehat{K}_i$ are identical by the ciphertext malleability of KEM. C_i gets $K'_i = \mathsf{KMaul}(K''_i, R_{\mathsf{A}} \oplus R''_{\mathsf{B}})$, $K'_j = \mathsf{KMaul}(K_j, R_{\mathsf{A}} \oplus R''_{\mathsf{B}})$, where $R_{\mathsf{A}} \oplus R''_{\mathsf{B}}$ is uniformly random. Therefore, K'_i and K'_j computed by C_i are unbiased.

Now we show that $u' = (C', c'_j, \overline{C}')$ and $(x', \overline{x}')$ output by F_i are unbiased. Let $(\hat{c}_j, \widehat{K}_j) \leftarrow\$ \mathsf{Encap}(pk_j)$. A specious C_i may send a biased $u = (C, c_j, \overline{C})$. However, this bias is always eliminated by an honest F_i. Upon receiving u from C_i, F_i sanitizes u using the $\mathsf{MaulCom}, \mathsf{Rerand}, \mathsf{MaulCom}', \mathsf{Rerand}', \mathsf{CMaul}$ and Rer algorithms. It is obvious that u' is unbiased: By the rerandomizability of Π_{Com} and Π'_{Com}, C' and $\overline{C}'$ are uniformly random. By the

perfect rerandomizability of KEM, c'_j has the same distribution as $\hat{c}_j$. Since Π_{Com} is malleable with respect to bKEM, it holds that $\widetilde{pk}' = \mathsf{mMaul}(\widetilde{pk}, r_0)$ is uniformly random over $\mathcal{PK}_{\mathsf{bKEM}}$. Moreover, by the tightness of Π_{Com}, for commitment C' and message $\widetilde{pk}'$, the opening message $x' = \mathsf{OpenRerand}(\mathsf{xMaul}(x, r_0), r_1)$ is unique. x' is also uniformly random due to the uniform randomness of $\widetilde{pk}'$. Analogously, since Π'_{Com} is malleable, it holds that $R'_{\mathsf{A}} = \mathsf{mMaul}(R_{\mathsf{A}}, r'_0 \oplus r_2^{-1})$ is uniformly random over $\mathcal{R}_{cm}$. By the tightness of Π'_{Com}, for $\overline{C}'$ and R'_{A}, the opening message $\overline{x}' = \mathsf{OpenRerand}(\mathsf{xMaul}(\overline{x}, r'_0 \oplus r_2^{-1}), r'_1)$ is unique and uniformly random.

The strong sanitation of F_j is proved similarly. First, an input-trigger attack against C_j is infeasible: $u'' = (C'', c''_j, \overline{C}'')$ and $(x'', \overline{x}'')$ received by C_j are always unbiased due to rerandomization via $\bar{r}_1, \bar{r}'_1$ and $\bar{\gamma}_j$. By the key malleability of bKEM, $\widetilde{pk}'' = \mathsf{KeyMaul}(\widetilde{pk}', \bar{r}_0)$ is uniformly random. Therefore, $\widetilde{K} = \mathsf{bKEM.Decap}(\widetilde{sk}'', \tilde{c}) = \mathsf{bKEM.Decap}(\widetilde{sk}', \tilde{c}')$ is also uniformly random where $\widetilde{sk}''$ is the corresponding private key of $\widetilde{pk}''$. The underlying key of c'_j is mauled into K''_j by $\bar{r}_2$. Since KEM is ciphertext-malleable, K''_j has the same distribution as $\widehat{K}_j$. C_j gets $K'_i = \mathsf{KMaul}(K_i, R''_{\mathsf{A}} \oplus R_{\mathsf{B}})$, $K'_j = \mathsf{KMaul}(K''_j, R''_{\mathsf{A}} \oplus R_{\mathsf{B}})$, where $R''_{\mathsf{A}} \oplus R_{\mathsf{B}}$ is uniformly random. Therefore, K'_i and K'_j computed by C_j are unbiased. Second, v' output by F_j is unbiased: F_j sanitizes $v = (\tilde{c}, c_i, R_{\mathsf{B}})$ (received from C_j) into $v' = (\tilde{c}', c'_i, R'_{\mathsf{B}})$ by computing $\tilde{c}' \leftarrow \mathsf{CKeyMaul}(\tilde{c}, \bar{r}_0)$, $c'_i \leftarrow \mathsf{Rer}(\mathsf{CMaul}(c_i, \bar{r}_2^{-1}), \bar{\gamma}_i)$ and $R'_{\mathsf{B}} = R_{\mathsf{B}} \oplus \bar{r}'_0$. Since $\widetilde{K} = \mathsf{bKEM.Decap}(\widetilde{sk}'', \tilde{c}) = \mathsf{bKEM.Decap}(\widetilde{sk}', \tilde{c}')$ is uniformly random, $\tilde{c}'$ is uniformly random (for a fixed $\widetilde{sk}'$). By the perfect rerandomizability of KEM, the distributions of c'_i and $\hat{c}_i$ are identical. R'_{B} is uniformly random due to the use of $\bar{r}'_0$.

□

Lemma 3. *The firewalls F_i and F_j are computationally transparent.*

Proof. We begin by analyzing the computational transparency of F_i. We show that for all environments $\mathcal{Z}$, it holds that $\text{EXEC}_{\mathsf{srAKE},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\mathsf{srAKE}',\mathcal{A},\mathcal{Z}}$, where $\mathcal{A}$ is the dummy adversary, srAKE and srAKE' run with dummy parties except for U, which is either $(\mathsf{C}_i, \mathsf{F}_i)$ or $(\mathsf{C}_i, \mathsf{ID})$. By the definition of computational transparency, $\mathsf{C}_i, \mathsf{F}_i$ and ID are uncorrupted. By Lemma 2, messages output by F_i and $K'_i, K'_j, \widetilde{K}$ computed by C_i are always unbiased, i.e., the distributions of the output of C_i in the presence of F_i are identical to that of the output of C_i in the presence of ID. Therefore, we have $\text{EXEC}_{\mathsf{srAKE},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\mathsf{srAKE}',\mathcal{A},\mathcal{Z}}$. The transparency of F_j can be proved in the same way.

□

We now present the simulation of an execution of srAKE in the corruption case where both the core and RF of one party are malicious, while the core and RF of the other party remain honest.

Lemma 4. *For every adversary $\mathcal{A}$ corrupting one party maliciously in an execution of srAKE in the random oracle model, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, it holds that $\text{EXEC}_{\mathsf{srAKE},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\mathsf{Wrap}(\mathcal{F}_{\mathsf{AKE}}),\mathcal{S},\mathcal{Z}}$.*

Proof sketch. We consider two sub-cases: either U or S is corrupt. If one of the parties is corrupt, $\mathcal{S}$ communicates with the corrupt party on behalf of the honest one. By the definition of NEWKEY interface of $\mathcal{F}_{\mathsf{AKE}}$, when the corrupt party interacts with an honest party, the session key is uniformly chosen by $\mathcal{F}_{\mathsf{AKE}}$. Therefore, $\mathcal{S}$ should ensure consistency between the specific RO answer and session key. Let K^*_i, K^*_j and $\widetilde{K}^*$ be the input keys of U to H. Let K^{**}_i, K^{**}_j and $\widetilde{K}^{**}$ be the input keys of S to H.

In the case of U-corruption, $\mathcal{S}$ simulates the behaviors of S. $\mathcal{A}$ is able to query $H(sid\|ssid\|\mathsf{aux}\|K_i^*\|K_j^*\|\widetilde{K}^*)$ upon receiving v' from $\mathcal{S}$. $\mathcal{S}$ is required to detect this RO query and answers with the session key. Therefore, upon receiving $u' = (C', c_j', \overline{C}')$ from $\mathcal{A}$, $\mathcal{S}$ should extract $(\widetilde{pk}', R_{\mathsf{A}}')$ from $(C', \overline{C}')$ and subsequently compute K_i^*, K_j^* and $\widetilde{K}^*$. Then before receiving x' and $\overline{x}'$, $\mathcal{S}$ is able to identify the RO query $H(sid\|ssid\|\mathsf{aux}\|K_i^*\|K_j^*\|\widetilde{K}^*)$ that corresponds to the session key. The extraction operation requires Π_{Com} to be extractable.

In the case of S-corruption, $\mathcal{A}$ should not be able to make an RO query $H(sid\|ssid\|\mathsf{aux}\|K_i^{**}\|K_j^{**}\|\widetilde{K}^{**})$ before receiving the opening $(x', \overline{x}')$ from $\mathcal{S}$. Otherwise, $\mathcal{S}$ cannot keep a consistency between the session key output by $\mathcal{F}_{\mathsf{AKE}}$ and that computed by $\mathcal{A}$. This is guaranteed by the computationally hiding property of Π_{Com}, which ensures that $\mathcal{A}$ cannot obtain any information about $\widetilde{pk}'$ from C'. The detailed proof is shown in Appendix D.3. □

It remains to prove that, when each party has an honest core and an honest RF, we can construct a simulator $\mathcal{S}$ (communicating with $\mathsf{Wrap}(\mathcal{F}_{\mathsf{AKE}})$) to simulate the execution of srAKE, such that $\mathcal{Z}$ can distinguish between the real-world protocol execution and the ideal-world process with negligible probability.

Lemma 5. *For every adversary $\mathcal{A}$ that does not corrupt any RFs or cores in an execution of srAKE in the random oracle model, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, it holds that $\text{EXEC}_{\mathsf{srAKE},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\mathsf{Wrap}(\mathcal{F}_{\mathsf{AKE}}),\mathcal{S},\mathcal{Z}}$.*

Proof sketch. The construction of $\mathcal{S}$ is shown in Fig. 10. In essence, the difference from $\mathcal{Z}$'s perspective in both executions lies in the computation of session keys: Parties in the real-world protocol execution compute the session key via an RO query. However, in the ideal world, $\mathcal{F}_{\mathsf{AKE}}$ would generate a uniformly random key without querying RO. We use a sequence of "hybrid games" to demonstrate the indistinguishability between the real world and the ideal world. The first game is the real-world execution where $\mathcal{Z}$ interacts with $\mathcal{A}$ and srAKE, and the last game is the ideal-world execution where $\mathcal{Z}$ interacts with $\mathcal{S}$ and $\mathsf{Wrap}(\mathcal{F}_{\mathsf{AKE}})$. We prove that the probability of $\mathcal{Z}$ outputting 1 in the first game is negligibly close to that in the last game. The detailed hybrid arguments are shown in Appendix D.4.

Simulator. Initially, $\mathcal{S}$ chooses public/private key pairs for both parties. $\mathcal{S}$ also runs KGen and KGen' to obtain the extraction keys ek and ek', respectively. Then $\mathcal{S}$ simulates C_i and F_i to generate $u' = (C', c_j', \overline{C}')$. When session $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$ receives $u'' = (C'', c_j'', \overline{C}'')$, $\mathcal{S}$ extracts $(\widetilde{pk}'', R_{\mathsf{A}}'')$ from $(C'', \overline{C}'')$, simulates C_j and F_j to generate v' and computes $K_i^{**}, K_j^{**}, \widetilde{K}^{**}$. Then $\mathcal{S}$ sends v' to $\mathcal{A}$. When session $\langle ssid, \mathsf{U}, \mathsf{S}\rangle$ receives $v'' = (\widetilde{c}'', c_i'', R_{\mathsf{B}}'')$, $\mathcal{S}$ computes $K_i^*, K_j^*, \widetilde{K}^*$ and simulates C_i and F_i to generate $(x', \overline{x}')$. If $(K_i^*, K_j^*, \widetilde{K}^*) \neq (K_i^{**}, K_j^{**}, \widetilde{K}^{**})$, i.e., $\mathcal{A}$ impersonates S, $\mathcal{S}$ sends an IMPERSONATE message to $\mathcal{F}_{\mathsf{AKE}}$. Moreover, if $u'' \neq u$ is generated adversarially (i.e., u'' is not produced in the manner of a firewall), which means $\mathcal{A}$ impersonates U, then $\mathcal{S}$ sends an IMPERSONATE message to $\mathcal{F}_{\mathsf{AKE}}$. $\mathcal{S}$ sends a NEWKEY message to $\mathcal{F}_{\mathsf{AKE}}$ and updates the session record if $\mathcal{F}_{\mathsf{AKE}}$ returns a session key. Then $\mathcal{S}$ sends $(x', \overline{x}')$ to $\mathcal{A}$. When $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$ receives $(x'', \overline{x}'')$, if it is invalid, $\mathcal{S}$ drops the message, else $\mathcal{S}$ sends a NEWKEY message to $\mathcal{F}_{\mathsf{AKE}}$ and updates the record if $\mathcal{F}_{\mathsf{AKE}}$ returns a session key. When a party $\mathcal{P}$ is COMPROMISED, $\mathcal{S}$ sends a COMPROMISE message to $\mathcal{F}_{\mathsf{AKE}}$, updates the session record if $\mathcal{F}_{\mathsf{AKE}}$ returns a key, and reveals the private key of $\mathcal{P}$ to $\mathcal{A}$.

$\mathcal{S}$ answers RO queries $(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3)$ by identifying those which correspond to potential session key computations of session $\langle ssid, \mathsf{U}, \mathsf{S}\rangle$ or $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$. If there exists a matching record in list L_H, $\mathcal{S}$ returns the key in that record. Else, $\mathcal{S}$ picks $\kappa^* \leftarrow\!\!\$\ \{0,1\}^w$. If there is a session record $\langle ssid, \mathsf{U}, \mathsf{S}, \kappa\rangle$, and it holds that $(K_1, K_2, K_3) = (K_i^*, K_j^*, \widetilde{K}^*)$ where K_i^*, K_j^* and $\widetilde{K}^*$ have been computed upon receiving v'', which implies that this RO query is a potential

Initially, $\mathcal{S}$ generates $(\rho, ek) \leftarrow\$ \mathsf{KGen}(1^\lambda)$, $(\rho', ek') \leftarrow\$ \mathsf{KGen}'(1^\lambda)$, $\mathsf{pp} \leftarrow \mathsf{bKEM.Setup}(1^\lambda), \mathsf{pp}' \leftarrow \mathsf{KEM.Setup}(1^\lambda)$. Then $\mathcal{S}$ generates $(pk_i, sk_i) \leftarrow\$ \mathsf{KEM.Gen}(\mathsf{pp}')$, $(pk_j, sk_j) \leftarrow\$ \mathsf{KEM.Gen}(\mathsf{pp}')$. $K_i^*, K_j^*, \widetilde{K}^*, K_i^{**}, K_j^{**}, \widetilde{K}^{**}$ and the list L_H are set to $\perp$.

On (SrvSession, $sid, ssid, \mathsf{U}, \mathsf{S}$) from $\mathcal{F}_{\mathsf{AKE}}$: Record $\langle ssid, \mathsf{U}, \mathsf{S}, \perp\rangle$. Generate the first message $u' = (C', c_j', \overline{C}')$ as U does in srAKE and send u' to $\mathcal{A}$ in the name of U.

On (SrvSession, $sid, ssid, \mathsf{U}, \mathsf{S}$) from $\mathcal{F}_{\mathsf{AKE}}$: Record $\langle ssid, \mathsf{S}, \mathsf{U}, \perp\rangle$.

On u'' from $\mathcal{A}$ to session $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$: \\ Receiving the 1st message.

- Parse $u'' = (C'', c_j'', \overline{C}'')$. Generate $\widetilde{pk}'' \leftarrow \mathsf{Extract}(ek, C'')$, $R_{\mathsf{A}}'' \leftarrow \mathsf{Extract}(ek, \overline{C}'')$, and compute $v', \widetilde{K}^{**}, K_i^{**}$ and K_j^{**} as S does.
- Send v' to $\mathcal{A}$ in the name of S.

On v'' from $\mathcal{A}$ to session $\langle ssid, \mathsf{U}, \mathsf{S}\rangle$: \\ Receiving the 2nd message.

- Compute $(x', \overline{x}')$, $\widetilde{K}^*, K_i^*$ and K_j^* as U does in srAKE.
- If $(K_i^*, K_j^*, \widetilde{K}^*) \neq (K_i^{**}, K_j^{**}, \widetilde{K}^{**})$, send (Impersonate, $sid, ssid, \mathsf{S}$) to $\mathcal{F}_{\mathsf{AKE}}$. Furthermore, if $\widetilde{pk}'' \neq \widetilde{pk}' \vee K_j'' \neq K_j' \vee R_{\mathsf{A}}'' \neq R_{\mathsf{A}}'$, send (Impersonate, $sid, ssid, \mathsf{U}$) to $\mathcal{F}_{\mathsf{AKE}}$.
- Pick $\kappa^* \leftarrow\$ \{0,1\}^w$ and send (NewKey, $sid, ssid, \mathsf{U}, \kappa^*$) to $\mathcal{F}_{\mathsf{AKE}}$. If $\mathcal{F}_{\mathsf{AKE}}$ returns κ, update record $\langle ssid, \mathsf{U}, \mathsf{S}, \perp\rangle$ to $\langle ssid, \mathsf{U}, \mathsf{S}, \kappa\rangle$.
- Send $(x', \overline{x}')$ to $\mathcal{A}$ in the name of U.

On $(x'', \overline{x}'')$ from $\mathcal{A}$ to session $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$: \\ Receiving the 3rd message.

- If $\mathsf{Verif}(\rho, C'', x'') = \perp \vee \mathsf{Verif}(\rho', \overline{C}'', \overline{x}'') = \perp$, drop the message. \\ x'' or $\overline{x}''$ is invalid.
- If there is a record $\langle ssid, \mathsf{S}, \mathsf{U}, \kappa\rangle$, instruct $\mathcal{F}_{\mathsf{AKE}}$ to send $(sid, ssid, \kappa)$ to S.
- If $\exists\langle ssid, \mathsf{S}, \mathsf{U}, \perp\rangle$, pick $\kappa^* \leftarrow\$ \{0,1\}^w$ and send (NewKey, $sid, ssid, \mathsf{S}, \kappa^*$) to $\mathcal{F}_{\mathsf{AKE}}$. If $\mathcal{F}_{\mathsf{AKE}}$ returns κ, update the record to $\langle ssid, \mathsf{S}, \mathsf{U}, \kappa\rangle$.

On (Compromise, $sid, \mathcal{P}$) from $\mathcal{A}$: \\ Compromise of a party $\mathcal{P}$.

- Send (Compromise, $sid, \mathcal{P}$) to $\mathcal{F}_{\mathsf{AKE}}$.
- If $\mathcal{F}_{\mathsf{AKE}}$ returns $(ssid, \mathcal{P}', \mathcal{P}, \kappa)$, update record $\langle ssid, \mathcal{P}', \mathcal{P}, \perp\rangle$ to $\langle ssid, \mathcal{P}', \mathcal{P}, \kappa\rangle$.
- Return the private key of $\mathcal{P}$ to $\mathcal{A}$.

On H query $(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3)$: \\ Answering RO queries.

- If $\exists\ (sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3, \kappa^*) \in L_H$, return κ^*. Else pick $\kappa^* \leftarrow\$ \{0,1\}^w$.
- If $\exists\langle ssid, \mathsf{U}, \mathsf{S}, \kappa\rangle$, and $(K_1, K_2, K_3) = (K_i^*, K_j^*, \widetilde{K}^*)$, or $\exists\langle ssid, \mathsf{S}, \mathsf{U}, \kappa\rangle$ and $(K_1, K_2, K_3) = (K_i^{**}, K_j^{**}, \widetilde{K}^{**})$, set $\kappa^* \leftarrow \kappa$.
- If $\exists\langle ssid, \mathsf{S}, \mathsf{U}, \perp\rangle$, U is compromised and $(K_1, K_2, K_3) = (K_i^{**}, K_j^{**}, \widetilde{K}^{**}) \wedge (\widetilde{pk}'' \neq \widetilde{pk}' \vee K_j'' \neq K_j' \vee R_{\mathsf{A}}'' \neq R_{\mathsf{A}}')$, pick $\kappa' \leftarrow\$ \{0,1\}^w$, send (Impersonate, $sid, ssid, \mathsf{U}$) and (NewKey, $sid, ssid, \mathsf{S}, \kappa'$) to $\mathcal{F}_{\mathsf{AKE}}$. If $\mathcal{F}_{\mathsf{AKE}}$ returns κ, update the record to $\langle ssid, \mathsf{S}, \mathsf{U}, \kappa\rangle$ and set $\kappa^* \leftarrow \kappa$. \\ $\mathcal{A}$ impersonates U and queries $H(sid\|ssid\|\mathsf{aux}\|K_i^{**}\|K_j^{**}\|\widetilde{K}^{**})$ upon receiving v'.

Add $(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3, \kappa^*)$ to L_H and output κ^*.

Figure 10: Simulator $\mathcal{S}$ for srAKE when both parties are honest.

computation of the key for session $\langle ssid, \mathsf{U}, \mathsf{S}\rangle$, $\mathcal{S}$ programs the session key to be κ. Symmetrically, if there is a session record $\langle ssid, \mathsf{S}, \mathsf{U}, \kappa\rangle$, and it holds that $(K_1, K_2, K_3) = (K_i^{**}, K_j^{**}, \widetilde{K}^{**})$ where K_i^{**}, K_j^{**} and $\widetilde{K}^{**}$ have been computed upon receiving u'', which implies that this RO query is a potential computation of the key for session $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$, $\mathcal{S}$ programs the session key to be κ. If $u'' \neq u'$ is generated adversarially, $\mathcal{A}$ compromises U and makes an RO query $(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3)$ before sending $(x'', \overline{x}'')$ such that $(K_1, K_2, K_3) = (K_i^{**}, K_j^{**}, \widetilde{K}^{**})$, then $\mathcal{S}$ sends IMPERSONATE and NEWKEY messages to $\mathcal{F}_{\mathsf{AKE}}$ but delays instructing $\mathcal{F}_{\mathsf{AKE}}$ to output the session key to S. Upon receiving κ from $\mathcal{F}_{\mathsf{AKE}}$, $\mathcal{S}$ responds the RO query with κ. □

By combining Lemmas 1 to 5 and the standard corruption translation table (Table 2), Theorem 1 follows immediately. □

7 Instantiations

Instantiation of bKEM. It is a straightforward construction of the DH-based KEM, denoted as DHK. The security of DHK is analyzed in Appendix E.2.2.

- $\mathsf{Setup}(1^\lambda)$: Randomly chooses a cyclic group $\mathbb{G}$ of order p with a generator g such that $2^\lambda \leq p < 2^{\lambda+1}$, sets $(\mathcal{K}, \mathcal{PK}, \mathcal{C}) := \{\mathbb{G}/1_{\mathbb{G}}\}^3, \mathcal{SK} := \mathbb{Z}_p^*$ and outputs $\mathsf{pp} = (\mathcal{K}, \mathcal{PK}, \mathcal{SK}, \mathcal{C}, p, g)$.
- $\mathsf{Gen}(\mathsf{pp})$: picks $k \leftarrow\!\$\ \mathbb{Z}_p^*$ and outputs $(sk, pk) := (k, g^k)$.
- $\mathsf{Encap}(pk)$: picks $\beta \leftarrow\!\$\ \mathbb{Z}_p^*$ and outputs $(c, K) := (g^\beta, pk^\beta)$.
- $\mathsf{Decap}(sk, c)$: outputs $K := c^{sk}$.
- $\mathsf{CKeyMaul}(c, r) := c^r$, $\mathsf{KeyMaul}(pk, r) := pk^r$.
- $\mathsf{Rg}(\mathsf{pp}) := r \leftarrow\!\$\ \mathbb{Z}_p^*$, $\mathsf{Kg}(pk, r) := pk^r$, $\mathsf{Cg}(r) := g^r$.

Instantiation of KEM. KEM can be instantiated with particular KEM derived from the ElGamal cryptosystem, called ElGamal-KEM. The security of ElGamal-KEM is analyzed in Appendix E.2.3.

- $\mathsf{Setup}(1^\lambda)$: Randomly chooses a cyclic group $\mathbb{G}$ of order p s.t. $2^\lambda \leq p < 2^{\lambda+1}$, sets $\mathcal{K} := \mathbb{G}, \mathcal{SK} := \mathbb{Z}_p, (\mathcal{PK}, \mathcal{C}) := \mathbb{G}^2$ and outputs $\mathsf{pp} = (\mathcal{K}, \mathcal{SK}, \mathcal{PK}, \mathcal{C}, p)$.
- $\mathsf{Gen}(\mathsf{pp})$: picks $k \leftarrow\!\$\ \mathbb{Z}_p, g \leftarrow\!\$\ \mathbb{G}$ and outputs $(sk, pk) := (k, (g, h = g^k))$.
- $\mathsf{Encap}(pk)$: parses $pk = (pk_1, pk_2)$, picks $\beta \leftarrow\!\$\ \mathbb{Z}_p, K \leftarrow\!\$\ \mathbb{G}$, computes $c \leftarrow (pk_1^\beta, pk_2^\beta \cdot K)$ and outputs (c, K).
- $\mathsf{Decap}(sk, c)$: parses $c = (c_1, c_2)$ and outputs $K := c_2/c_1^{sk}$.
- $\mathsf{CMaul}(c, r)$: parses $c = (c_1, c_2)$, outputs $c' = (c_1, c_2 \cdot r)$.
- $\mathsf{KMaul}(K, r)$: outputs $K \cdot r$.
- $\mathsf{Rer}(c, r)$: parses $c = (c_1, c_2)$, outputs $c' = (c_1 \cdot g^r, c_2 \cdot h^r)$.

Instantiations of Π_{Com} and Π'_{Com}. Π_{Com} (see Table 3) is instantiated by ElGamal encryption. Π'_{Com} is the same as Π_{Com} except the following algorithms:

- $\mathsf{MaulCom}'(C, r)$: parse $C = (C_1, C_2)$, return $(C'_1, C'_2) := (C_1, C_2 \cdot r)$. $(\mathcal{R}'_{mcom} = \mathbb{G}.)$
- $\mathsf{mMaul}'(m, r)$: return $m' := m \cdot r$. $\mathsf{xMaul}'(x, r)$: return x.

We defer the security analysis of Π_{Com} and Π'_{Com} to Appendix E.2.1. The instantiation of cores and RFs is postponed to Appendix E.1.

References

[1] Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) Topics in Cryptology – CT-RSA 2001. Lecture

Table 3: Instantiation of Π_{Com}.

$\mathsf{KGen}(1^\lambda)$: Randomly choose a cyclic group $\mathbb{G}$ of order p with a generator g such that $2^\lambda \leq p < 2^{\lambda+1}$. Set $\mathcal{M}_{com} := \mathbb{G}, (\mathcal{R}_{com}, \mathcal{R}_{mcom}) := \mathbb{Z}_p^2$, $\mathcal{C}_{com} := \mathbb{G}^2$. Then randomly pick $t \leftarrow\!\!\$\ \mathbb{Z}_p$, compute $h \leftarrow g^t$ as the CRS and output $\rho := (\mathcal{M}_{com}, \mathcal{R}_{com}, \mathcal{R}_{mcom}, \mathcal{C}_{com}, g, h)$ and $ek := t$.			
$\mathsf{Extract}(C, ek)$: parse $C = (C_1, C_2)$. output $m \leftarrow C_2/(C_1)^{ek}$. $\mathsf{Com}(m, \rho; y)$: output $C := (g^y, h^y \cdot m)$. $\mathsf{Open}(\rho, C, y)$: output y.	$\mathsf{Verif}(\rho, C, y)$: parse $C = (C_1, C_2)$, if $C_1 = g^y$, output $m := C_2/h^y$; otherwise, output $\perp$.	$\mathsf{Rerand}(C, r)$: parse $C = (C_1, C_2)$, compute $(C_1', C_2') \leftarrow (C_1 \cdot g^r, C_2 \cdot h^r)$. $\mathsf{OpenRerand}(x, r)$: compute $x' := (x + r)$ and return x'.	$\mathsf{MaulCom}(C, r)$: parse $C = (C_1, C_2)$, return $(C_1', C_2') := (C_1^r, C_2^r)$. $\mathsf{mMaul}(m, r)$: output $m' := m^r$. $\mathsf{xMaul}(x, r)$: return $x \cdot r$.

Notes in Computer Science, vol. 2020, pp. 143–158. Springer, Heidelberg, Germany, San Francisco, CA, USA (Apr 8–12, 2001). https://doi.org/10.1007/3-540-45353-9_12

[2] Armour, M., Poettering, B.: Algorithm substitution attacks against receivers. Int. J. Inf. Sec. **21**(5), 1027–1050 (2022)

[3] Arnold, P., Berndt, S., Müller-Quade, J., Ottenhues, A.: Protection against subversion corruptions via reverse firewalls in the plain universal composability framework. In: Fischlin, M., Moonsamy, V. (eds.) Applied Cryptography and Network Security. pp. 510–539. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-95764-2_20

[4] Arnold, P., Berndt, S., Müller-Quade, J., Ottenhues, A.: Protection against subversion corruptions via reverse firewalls in the plain universal composability framework. Cryptology ePrint Archive, Paper 2023/1951 (2023). https://doi.org/10.1007/978-3-031-95764-2_20, https://eprint.iacr.org/2023/1951

[5] Ateniese, G., Magri, B., Venturi, D.: Subversion-resilient signature schemes. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015: 22nd Conference on Computer and Communications Security. pp. 364–375. ACM Press, Denver, CO, USA (Oct 12–16, 2015). https://doi.org/10.1145/2810103.2813635

[6] Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure computation without authentication. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 361–377. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005). https://doi.org/10.1007/11535218_22

[7] Bellare, M., Canetti, R., Krawczyk, H.: A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In: 30th Annual ACM Symposium on Theory of Computing. pp. 419–428. ACM Press, Dallas, TX, USA (May 23–26, 1998). https://doi.org/10.1145/276698.276854

[8] Bellare, M., Jaeger, J., Kane, D.: Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015: 22nd Conference on Computer and Communications Security. pp. 1431–1440. ACM Press, Denver, CO, USA (Oct 12–16, 2015). https://doi.org/10.1145/2810103.2813681

[9] Bellare, M., Paterson, K.G., Rogaway, P.: Security of symmetric encryption against mass surveillance. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 1–19. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014). https://doi.org/10.1007/978-3-662-44371-2_1

[10] Bemmann, P., Berndt, S., Chen, R.: Subversion-resilient signatures without random oracles. In: ACNS 24: 22nd International Conference on Applied Cryptography and Network Security, Part I. pp. 351–375. Lecture Notes in Computer Science, Springer, Cham, Switzerland (Jun 21–24, 2024). https://doi.org/10.1007/978-3-031-54770-6_14

[11] Bemmann, P., Berndt, S., Diemert, D., Eisenbarth, T., Jager, T.: Subversion-resilient authenticated encryption without random oracles. In: Tibouchi, M., Wang, X. (eds.) ACNS 23: 21st International Conference on Applied Cryptography and Network Security, Part II. Lecture Notes in Computer Science, vol. 13906, pp. 460–483. Springer, Heidelberg, Germany, Kyoto, Japan (Jun 19–22, 2023). https://doi.org/10.1007/978-3-031-33491-7_17

[12] Bemmann, P., Chen, R., Jager, T.: Subversion-resilient public key encryption with practical watchdogs. In: Garay, J. (ed.) PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 12710, pp. 627–658. Springer, Heidelberg, Germany, Virtual Event (May 10–13, 2021). https://doi.org/10.1007/978-3-030-75245-3_23

[13] Berndt, S., Liskiewicz, M.: Algorithm substitution attacks from a steganographic perspective. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security. pp. 1649–1660. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017). https://doi.org/10.1145/3133956.3133981

[14] Berndt, S., Wichelmann, J., Pott, C., Traving, T.H., Eisenbarth, T.: ASAP: Algorithm substitution attacks on cryptographic protocols. In: Suga, Y., Sakurai, K., Ding, X., Sako, K. (eds.) ASIACCS 22: 17th ACM Symposium on Information, Computer and Communications Security. pp. 712–726. ACM Press, Nagasaki, Japan (May 30 – Jun 3, 2022). https://doi.org/10.1145/3488932.3517387

[15] Bossuat, A., Bultel, X., Fouque, P.A., Onete, C., van der Merwe, T.: Designing reverse firewalls for the real world. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020: 25th European Symposium on Research in Computer Security, Part I. Lecture Notes in Computer Science, vol. 12308, pp. 193–213. Springer, Heidelberg, Germany, Guildford, UK (Sep 14–18, 2020). https://doi.org/10.1007/978-3-030-58951-6_10

[16] Canetti, R.: Universally composable security. J. ACM **67**(5), 28:1–28:94 (2020)

[17] Canetti, R., Jain, P., Swanberg, M., Varia, M.: Universally composable end-to-end secure messaging. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022, Part II. Lecture Notes in Computer Science, vol. 13508, pp. 3–33. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 15–18, 2022). https://doi.org/10.1007/978-3-031-15979-4_1

[18] Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: Knudsen, L.R. (ed.) Advances in Cryptology – EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pp. 337–351. Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Apr 28 – May 2, 2002). https://doi.org/10.1007/3-540-46035-7_22

[19] Chakraborty, S., Dziembowski, S., Nielsen, J.B.: Reverse firewalls for actively secure MPCs. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part II. Lecture Notes in Computer Science, vol. 12171, pp. 732–762. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56880-1_26

[20] Chakraborty, S., Ganesh, C., Pancholi, M., Sarkar, P.: Reverse firewalls for adaptively secure MPC without setup. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021, Part II. Lecture Notes in Computer Science, vol. 13091, pp. 335–364. Springer, Heidelberg, Germany, Singapore (Dec 6–10, 2021). https://doi.org/10.1007/978-3-030-92075-3_12

[21] Chakraborty, S., Ganesh, C., Sarkar, P.: Reverse firewalls for oblivious transfer extension and applications to zero-knowledge. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part I. Lecture Notes in Computer Science, vol. 14004, pp. 239–270. Springer, Heidelberg, Germany, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30545-0_9

[22] Chakraborty, S., Magliocco, L., Magri, B., Venturi, D.: Key exchange in the post-snowden era: Universally composable subversion-resilient PAKE. Cryptology ePrint Archive, Paper 2023/1827 (2023), https://eprint.iacr.org/2023/1827

[23] Chakraborty, S., Magliocco, L., Magri, B., Venturi, D.: Key exchange in the post-snowden era: Universally composable subversion-resilient PAKE. In: Advances in Cryptology – ASIACRYPT 2024, Part V. Lecture Notes in Computer Science, vol. 15488, pp. 101–133. Springer, Singapore, Singapore (Dec 7–11, 2024). https://doi.org/10.1007/978-981-96-0935-2_4

[24] Chakraborty, S., Magri, B., Nielsen, J.B., Venturi, D.: Universally composable subversion-resilient cryptography. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part I. Lecture Notes in Computer Science, vol. 13275, pp. 272–302. Springer, Heidelberg, Germany, Trondheim, Norway (May 30 – Jun 3, 2022). https://doi.org/10.1007/978-3-031-06944-4_10

[25] Chen, R., Huang, X., Yung, M.: Subvert KEM to break DEM: Practical algorithm-substitution attacks on public-key encryption. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020, Part II. Lecture Notes in Computer Science, vol. 12492, pp. 98–128. Springer, Heidelberg, Germany, Daejeon, South Korea (Dec 7–11, 2020). https://doi.org/10.1007/978-3-030-64834-3_4

[26] Chen, R., Mu, Y., Yang, G., Susilo, W., Guo, F., Zhang, M.: Cryptographic reverse firewall via malleable smooth projective hash functions. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 844–876. Springer, Heidelberg, Germany, Hanoi, Vietnam (Dec 4–8, 2016). https://doi.org/10.1007/978-3-662-53887-6_31

[27] Chow, S.S.M., Russell, A., Tang, Q., Yung, M., Zhao, Y., Zhou, H.S.: Let a non-barking watchdog bite: Cliptographic signatures with an offline watchdog. In: Lin, D., Sako, K. (eds.) PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 11442, pp. 221–251. Springer, Heidelberg, Germany, Beijing, China (Apr 14–17, 2019). https://doi.org/10.1007/978-3-030-17253-4_8

[28] Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. Journal of Cryptology **33**(4), 1914–1983 (Oct 2020). https://doi.org/10.1007/s00145-020-09360-1

[29] Cohn-Gordon, K., Cremers, C., Gjøsteen, K., Jacobsen, H., Jager, T.: Highly efficient key exchange protocols with optimal tightness. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 767–797. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019). https://doi.org/10.1007/978-3-030-26954-8_25

[30] Degabriele, J.P., Farshim, P., Poettering, B.: A more cautious approach to security against mass surveillance. In: Leander, G. (ed.) Fast Software Encryption – FSE 2015. Lecture Notes in Computer Science, vol. 9054, pp. 579–598. Springer, Heidelberg, Germany, Istanbul, Turkey (Mar 8–11, 2015). https://doi.org/10.1007/978-3-662-48116-5_28

[31] Dodis, Y., Mironov, I., Stephens-Davidowitz, N.: Message transmission with reverse firewalls—secure communication on corrupted machines. Cryptology ePrint Archive, Paper 2015/548 (2015), https://eprint.iacr.org/2015/548

[32] Dodis, Y., Mironov, I., Stephens-Davidowitz, N.: Message transmission with reverse firewalls—secure communication on corrupted machines. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 341–372. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016). https://doi.org/10.1007/978-3-662-53018-4_13

[33] Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol. Journal of Cryptology **34**(4), 37 (Oct 2021). https://doi.org/10.1007/s00145-021-09384-1

[34] Faonio, A., Fiore, D., Herranz, J., Ràfols, C.: Structure-preserving and re-randomizable RCCA-secure public key encryption and its applications. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019, Part III. Lecture Notes in Computer Science, vol. 11923, pp. 159–190. Springer, Cham, Switzerland, Kobe, Japan (Dec 8–12, 2019). https://doi.org/10.1007/978-3-030-34618-8_6

[35] Fischlin, M., Mazaheri, S.: Self-guarding cryptographic protocols against algorithm substitution attacks. In: Chong, S., Delaune, S. (eds.) CSF 2018: IEEE 31st Computer Security Foundations Symposium. pp. 76–90. IEEE Computer Society Press, Oxford, UK (Jul 9–12, 2018). https://doi.org/10.1109/CSF.2018.00013

[36] Galteland, H., Gjøsteen, K.: Subliminal channels in post-quantum digital signature schemes. Cryptology ePrint Archive, Report 2019/574 (2019), https://eprint.iacr.org/2019/574

[37] Ganesh, C., Magri, B., Venturi, D.: Cryptographic reverse firewalls for interactive proof systems. In: Czumaj, A., Dawar, A., Merelli, E. (eds.) ICALP 2020: 47th International Colloquium on Automata, Languages and Programming. LIPIcs, vol. 168, pp. 55:1–55:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Saarbrücken, Germany (Jul 8–11, 2020). https://doi.org/10.4230/LIPIcs.ICALP.2020.55

[38] Gellert, K., Gjøsteen, K., Jacobsen, H., Jager, T.: On optimal tightness for key exchange with full forward secrecy via key confirmation. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part IV. Lecture Notes in Computer Science, vol.

14084, pp. 297–329. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38551-3_10

[39] Gu, Y., Jarecki, S., Krawczyk, H.: KHAPE: Asymmetric PAKE from key-hiding key exchange. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part IV. Lecture Notes in Computer Science, vol. 12828, pp. 701–730. Springer, Heidelberg, Germany, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84259-8_24

[40] Gu, Y., Jarecki, S., Krawczyk, H.: KHAPE: Asymmetric PAKE from key-hiding key exchange. Cryptology ePrint Archive, Paper 2021/873 (2021), https://eprint.iacr.org/2021/873

[41] Hofheinz, D., Müller-Quade, J., Steinwandt, R.: Initiator-resilient universally composable key exchange. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003: 8th European Symposium on Research in Computer Security. Lecture Notes in Computer Science, vol. 2808, pp. 61–84. Springer, Heidelberg, Germany, Gjøvik, Norway (Oct 13–15, 2003). https://doi.org/10.1007/978-3-540-39650-5_4

[42] Hülsing, A., Ning, K.C., Schwabe, P., Weber, F., Zimmermann, P.R.: Post-quantum WireGuard. In: 2021 IEEE Symposium on Security and Privacy. pp. 304–321. IEEE Computer Society Press, San Francisco, CA, USA (May 24–27, 2021). https://doi.org/10.1109/SP40001.2021.00030

[43] Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 456–486. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018). https://doi.org/10.1007/978-3-319-78372-7_15

[44] Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 546–566. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005). https://doi.org/10.1007/11535218_33

[45] Liu, C., Chen, R., Wang, Y., Wang, Y.: Asymmetric subversion attacks on signature schemes. In: Susilo, W., Yang, G. (eds.) ACISP 18: 23rd Australasian Conference on Information Security and Privacy. Lecture Notes in Computer Science, vol. 10946, pp. 376–395. Springer, Heidelberg, Germany, Wollongong, NSW, Australia (Jul 11–13, 2018). https://doi.org/10.1007/978-3-319-93638-3_22

[46] Liu, J., Chen, R., Wang, Y., Tang, X., Su, J.: Subversion-resilient authenticated key exchange with reverse firewalls. In: Liu, J.K., Chen, L., Sun, S.F., Liu, X. (eds.) Provable and Practical Security. pp. 181–200. Springer Nature Singapore, Singapore (2025). https://doi.org/10.1007/978-981-96-0957-4_10

[47] Liu, J., Wang, Y., Chen, R., Tang, X., Su, J.: srCPace: Universally composable PAKE with subversion-resilience. In: Lin, D., Wang, M., Yung, M. (eds.) Information Security and Cryptology. pp. 211–231. Springer Nature Singapore, Singapore (2025). https://doi.org/10.1007/978-981-96-4731-6_11

[48] Liu, J., Wang, Y., Tang, X., Chen, R., Huang, X., Su, J.: srTLS: Secure TLS handshake on corrupted machines. IEEE Trans. Dependable Secur. Comput. **22**(3), 3137–3154 (2025). https://doi.org/10.1109/TDSC.2024.3524681

[49] Mironov, I., Stephens-Davidowitz, N.: Cryptographic reverse firewalls. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part II. Lecture Notes in Computer Science, vol. 9057, pp. 657–686. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46803-6_22

[50] Perlroth, N., Larson, J., Shane, S.: Secret documents reveal nsa campaign against encryption (2013), https://archive.nytimes.com/www.nytimes.com/interactive/2013/09/05/us/documents-reveal-nsa-campaign-against-encryption.html

[51] Russell, A., Tang, Q., Yung, M., Zhou, H.S.: Cliptography: Clipping the power of kleptographic attacks. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 10032, pp. 34–64. Springer, Heidelberg, Germany, Hanoi, Vietnam (Dec 4–8, 2016). https://doi.org/10.1007/978-3-662-53890-6_2

[52] Russell, A., Tang, Q., Yung, M., Zhou, H.S.: Generic semantic security against a kleptographic adversary. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security. pp. 907–922. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017). https://doi.org/10.1145/3133956.3133993

[53] Russell, A., Tang, Q., Yung, M., Zhou, H.S.: Correcting subverted random oracles. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 241–271. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018). https://doi.org/10.1007/978-3-319-96881-0_9

[54] Santos, B.F.D., Gu, Y., Jarecki, S., Krawczyk, H.: Asymmetric PAKE with low computation and communication. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part II. Lecture Notes in Computer Science, vol. 13276, pp. 127–156. Springer, Heidelberg, Germany, Trondheim, Norway (May 30 – Jun 3, 2022). https://doi.org/10.1007/978-3-031-07085-3_5

[55] Shoup, V.: A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112 (2001), https://eprint.iacr.org/2001/112

[56] Steinfeld, R., Baek, J., Zheng, Y.: On the necessity of strong assumptions for the security of a class of asymmetric encryption schemes. In: Batten, L.M., Seberry, J. (eds.) ACISP 02: 7th Australasian Conference on Information Security and Privacy. Lecture Notes in Computer Science, vol. 2384, pp. 241–256. Springer, Heidelberg, Germany, Melbourne, Victoria, Australia (Jul 3–5, 2002). https://doi.org/10.1007/3-540-45450-0_20

[57] Teseleanu, G.: Threshold kleptographic attacks on discrete logarithm based signatures. In: Lange, T., Dunkelman, O. (eds.) Progress in Cryptology - LATINCRYPT 2017: 5th International Conference on Cryptology and Information Security in Latin America. Lecture Notes in Computer Science, vol. 11368, pp. 401–414. Springer, Heidelberg, Germany, Havana, Cuba (Sep 20–22, 2019). https://doi.org/10.1007/978-3-030-25283-0_21

[58] Young, A., Yung, M.: The dark side of "black-box" cryptography, or: Should we trust capstone? In: Koblitz, N. (ed.) Advances in Cryptology – CRYPTO'96. Lecture Notes in Computer Science, vol. 1109, pp. 89–103. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 1996). https://doi.org/10.1007/3-540-68697-5_8

[59] Young, A., Yung, M.: Kleptography: Using cryptography against cryptography. In: Fumy, W. (ed.) Advances in Cryptology – EUROCRYPT'97. Lecture Notes in Computer Science, vol. 1233, pp. 62–74. Springer, Heidelberg, Germany, Konstanz, Germany (May 11–15, 1997). https://doi.org/10.1007/3-540-69053-0_6

A Basic Definitions

A.1 Diffie-Hellman Assumptions

Definition 16 (Decisional Diffie-Hellman (DDH) assumption)**.** *Let $\mathbb{G}$ be a cyclic group of order p with a generator g such that $2^\lambda \leq p < 2^{\lambda+1}$. Let $(x, y) \leftarrow\!\!\$\ \mathbb{Z}_p^2$, $z_0 = xy, z_1 \leftarrow\!\!\$\ \mathbb{Z}_p$. The advantage of an adversary $\mathcal{A}$ in solving DDH problem for $\mathbb{G}$ is*

$$\mathrm{DDHAdv}[\mathcal{A}, \mathbb{G}](\lambda) \coloneqq |\Pr[\mathcal{A}(g, g^x, g^y, g^{z_0}) = 1] - \Pr[\mathcal{A}(g, g^x, g^y, g^{z_1}) = 1]|.$$

The DDH assumption holds for $\mathbb{G}$ if for all PPT adversaries $\mathcal{A}$, it holds that $\mathrm{DDHAdv}[\mathcal{A}, \mathbb{G}](\lambda) \leq \mathsf{negl}(\lambda)$.

Given a cyclic group $\mathbb{G}$ of order p with a generator g and a fixed $x \in \mathbb{Z}_p$, $\mathcal{O}_{g^x}(\cdot, \cdot)$ denotes a restricted decisional DH oracle that takes as input $(Y, Z) \in \mathbb{G}^2$, and returns 1 if $Y^x = Z$, otherwise returns 0.

Definition 17 (Strong Diffie-Hellman (SDH) assumption [1])**.** *Let $\mathbb{G}$ be a cyclic group of order p with a generator g such that $2^\lambda \leq p < 2^{\lambda+1}$. Then the advantage of an adversary $\mathcal{A}$ in solving the SDH problem for $\mathbb{G}$ is defined as*

$$\mathrm{Adv}_{\mathbb{G}}^{\mathsf{SDH}}(\mathcal{A}, \lambda) \coloneqq \Pr[(x, y) \leftarrow\!\!\$\ \mathbb{Z}_p^2 : \mathcal{A}^{\mathcal{O}_{g^x}(\cdot,\cdot)}(g, g^x, g^y) = g^{xy}].$$

The SDH assumption for $\mathbb{G}$ holds if for all PPT adversaries $\mathcal{A}$, the advantage $\mathrm{Adv}_{\mathbb{G}}^{\mathsf{SDH}}(\mathcal{A}, \lambda) \leq \mathsf{negl}(\lambda)$.

The inverse CDH (iCDH) assumption is a variant of the computational Diffie-Hellman assumption. First, we define a $\mathtt{DH}$ function: Given a cyclic group $\mathbb{G}$ of order p with a generator g, let $a, b, c \leftarrow \mathbb{Z}_p$, $u = g^a, v = g^b$ and $w = g^c$. $\mathtt{DH}(u, v, w)$ returns 1 if $c = a \cdot b$, otherwise returns 0.

Definition 18 (Inverse Computational Diffie-Hellman (iCDH) assumption [31])**.** *Let $\mathbb{G}$ be a cyclic group of order p with a generator g such that $2^\lambda \leq p < 2^{\lambda+1}$. Then the advantage of an adversary $\mathcal{A}$ in solving the inverse CDH problem for $\mathbb{G}$ is defined as* $\mathrm{Adv}_{\mathbb{G}}^{\mathsf{iCDH}}(\mathcal{A}, \lambda) \coloneqq \Pr_{(x,y)\leftarrow\!\!\$\mathbb{Z}_p^2}[\mathtt{DH}(u, g^x, h) = 1 \wedge \mathtt{DH}(v, g^y, h) = 1 | (h, u, v) \leftarrow \mathcal{A}(g, g^x, g^y)]$ *where $h \in \mathbb{G}/\{1_{\mathbb{G}}\}$. The iCDH assumption for $\mathbb{G}$ holds if for all PPT adversaries $\mathcal{A}$, we have* $\mathrm{Adv}_{\mathbb{G}}^{\mathsf{iCDH}}(\mathcal{A}, \lambda) \leq \mathsf{negl}(\lambda)$.

A.2 Indistinguishable Distribution Ensembles

Let us recall the definition of indistinguishable distribution ensembles [16]: Firstly, a *probability distribution ensemble* $X = \{X(k, r)\}_{k\in\mathbb{N}, r\in\{0,1\}^*}$ is defined as an infinite set of probability distributions, where a distribution $X(k, r)$ is associated with each $k \in \mathbb{N}$ and $r \in \{0, 1\}^*$. In the context of protocol execution, these ensembles describe the outcomes of computations, where k represents the security parameter, and r represents the input parameter.

Definition 19 (Indistinguishable distribution ensembles [16])**.** *Two binary probability distribution ensembles X and Y are* indistinguishable *(written as $X \approx Y$) if for any $c, d \in \mathbb{N}$, there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and all $r \in \cup_{\kappa\leq k^d}\{0, 1\}^\kappa$ we have:*

$$|\Pr[X(k, r) = 1] - \Pr[Y(k, r) = 1]| < k^{-c}.$$

A.3 Properties for Commitment Scheme

Definition 20 (Perfectly binding [20]). *A commitment scheme* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$ *is* perfectly binding *if for any unbounded adversary* $\mathcal{A}$*, we have*

$$\Pr[(m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(\rho) | \mathsf{Com}(m_0, \rho; r_0) = \mathsf{Com}(m_1, \rho; r_1)] = 0,$$

where $\rho \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$.

Definition 21 (Computationally hiding [20]). *A commitment scheme* $\Pi_{\mathsf{Com}} = (\mathsf{KGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verif})$ *is* computationally hiding *if for all PPT adversaries* $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$*, the probability that the hiding experiment in Fig. 11 outputs 1 is bounded by* $\frac{1}{2} + \mathsf{negl}(\lambda)$.

Exp $\mathrm{Hiding}(\lambda, \Pi_{\mathsf{Com}}, \mathcal{A})$

$\rho \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$
$(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\rho)$
$r \leftarrow\!\!\$\ \mathcal{R}_{com}; C \leftarrow \mathsf{Com}(m_b, \rho; r)$
$b' \leftarrow \mathcal{A}_2(C, \mathsf{st})$
if $b = b'$
  return 1
else
  return 0

Figure 11: The hiding experiment for the adversary $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ against Π_{Com}, where $\mathcal{R}_{com}$ is the randomness space.

B Definitions in the srUC Model [24]

In this part, we recall the definition of specious corruptions, specious environments and specious adversaries in [24]. The specious corruption is defined through testing. Consider a test T that is given non-rewinding black-box access to either $\widetilde{\mathsf{C}}_i$ or C_i and attempts to guess which one it interacts with. A subversion is deemed *specious* if it withstands all efficient tests. It is assumed that the test does not have access to the random choices made by $\widetilde{\mathsf{C}}_i$. The subverter is allowed to possess some auxiliary information about the subversion and its use of randomness, enabling covert communication with the subverted party while evading detection by any test.

More specifically, consider a machine T and an interactive machine $\widetilde{\mathsf{C}}$. Let $\mathsf{T}^{\widetilde{\mathsf{C}}}$ indicate that T has non-rewinding black-box access to $\widetilde{\mathsf{C}}$. During the execution of $\mathsf{T}^{\widetilde{\mathsf{C}}}$, if $\widetilde{\mathsf{C}}$ requests a random bit, a uniformly random bit is sampled and given to $\widetilde{\mathsf{C}}$ without being revealed to T. The game for a test T and an efficiently sampleable distribution $\mathcal{D}$ is defined as follows.

- Sample $(\widetilde{\mathsf{C}}, a) \leftarrow \mathsf{D}$, where a is an auxiliary string (i.e., the auxiliary information shared by the subverter and the subverted party).
- Pick $b \leftarrow\!\!\$\ \{0,1\}$. If $b = 0$, then run $\mathsf{T}^{\widetilde{\mathsf{C}}}$ and obtain a guess $g \in \{0,1\}$. Else run T^{C} and obtain a guess $g \in \{0,1\}$.
- Output $c = b \oplus g$.

Let $\text{TEST}_{\mathsf{D},\mathsf{T}} := \Pr[c = 0]$, which refers to the probability that the guess of b is correct. Based on this game, a formal definition of *specious subversion* is given below.

Definition 22 (Specious subversion [24])**.** *We say that* D *is* computationally specious *if for all PPT tests* T *it holds that* $\text{TEST}_{\mathsf{D},\mathsf{T}} - \frac{1}{2}$ *is negligible.*

Furtherly, speciousness for a set of subversions is defined as follows.

Definition 23 (Specious subversions [24])**.** *Let* D *be an efficiently sampleable distribution with outputs of the form* $(\widetilde{\mathsf{C}}_1, \cdots, \widetilde{\mathsf{C}}_m, a) \leftarrow \mathsf{D}$*. Let* D_i *be the distribution sampling* $(\widetilde{\mathsf{C}}_1, \cdots, \widetilde{\mathsf{C}}_m, a) \leftarrow \mathsf{D}$ *and then outputting* $(\widetilde{\mathsf{C}}_i, (i, a))$*.* D *is specious if each* D_i *is specious.*

Definition 24 (Specious corruption [24])**.** *We say that a party accepts* specious corruptions *if, upon receiving input* $(\text{SPECIOUS}, \widetilde{\mathsf{C}})$ *from the adversary, it replaces its code with* $\widetilde{\mathsf{C}}$*. If the input* $(\text{SPECIOUS}, \widetilde{\mathsf{C}})$ *is not the first one received by the party, then it is ignored.*

In addition, we recall the concept of a "specious environment" that generates specious corruptions, and a "specious adversary" who does specious corruptions received from $\mathcal{Z}$. Both are defined as follows [24].

Definition 25 (Specious environment [24])**.** *An environment* $\mathcal{Z}$ *is said to prepare specious corruptions if it behaves as follows:*

- *First, it writes* $(\text{SPECIOUS}, \mathsf{D})$ *on a special tape, where* D *is specious.*
- *Then, it picks* $(\widetilde{\mathsf{C}}_1, \cdots, \widetilde{\mathsf{C}}_m, a) \leftarrow\!\!\$\ \mathsf{D}$ *and writes this on the special tape as well.*
- *Finally, it inputs* $(\text{SPECIOUS}, \widetilde{\mathsf{C}}_1, \cdots, \widetilde{\mathsf{C}}_m)$ *to the adversary.*

We say that such an environment is specious*.*

Definition 26 (Specious adversary [24])**.** *We say that an adversary interacting with a specious environment does specious corruptions if whenever the adversary inputs* $(\text{SPECIOUS}, \widetilde{\mathsf{C}})$ *to a party, then* $\widetilde{\mathsf{C}}$ *is from the list* $(\text{SPECIOUS}, \widetilde{\mathsf{C}}_1, \cdots, \widetilde{\mathsf{C}}_m)$ *received from the specious environment. We call such an adversary* specious*. An adversary that never inputs* $(\text{SPECIOUS}, \widetilde{\mathsf{C}})$ *to any party is also called specious.*

C Proof of Lemma 1

Proof. According to the definition of strong sanitation, a specious core hidden behind an honest RF is indistinguishable from an honest core behind the same RF. Consequently, we can eliminate the case of a specious core and an honest RF. In specific, assume that Π' is a modified version of Π where core C_i is replaced by its incorruptible version $\widehat{\mathsf{C}}_i$. By the definition of strong sanitation, we have $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\Pi',\mathcal{A},\mathcal{Z}}$. Here we replace $\mathcal{Z}$ by $\mathcal{Z}'$ that does not forward the specious corruption command to the core $\widehat{\mathsf{C}}_i$, as forwarding specious corruption command to an incorruptible core or not does not make any difference. Then, we replace $\widehat{\mathsf{C}}_i$ by C_i, as there is no specious corruption to omit. Therefore, the simulation of a specious core and an honest RF is indistinguishable from that of an honest core and an honest RF. Note that the case of an honest core and an honest RF is a special case of a specious core and an honest RF.

We show that if all firewalls are computationally transparent, the analysis of the case involving an honest core and a semi-honest firewall can be reduced to that of an honest core and an honest firewall: Due to the computational transparency, we can substitute an RF with an identity RF denoted by ID. Furthermore, the only impact of a semi-honest corruption of

ID is to reveal any messages transmitted through it and the values shared with it during the initialization phase. However, in the case of an honest identity firewall, these values are already exposed to the adversary. Thus, a semi-honest ID does not yield any additional information beyond what an honest ID would. □

D Omitted Description and Security Analysis of srAKE

D.1 A Textual Description of srAKE

A textual description of srAKE is depicted in Fig. 12. The initialization phase is identical to that described in Section 6.2 and is thus omitted in Fig. 12.

D.2 Correctness Analysis

Correctness. Now we show that C_i and C_j compute the same session key K. Specifically, both of them compute the same K'_i, K'_j and $\widetilde{K}$. In the following analysis, we assume that all entities, including $\mathsf{C}_i, \mathsf{F}_i, \mathsf{F}_j$ and C_j are honest.

Computing K'_i. Upon receiving c''_i, C_i obtains K'_i by computing $K''_i \leftarrow \mathsf{KEM.Decap}(sk_1, c''_i)$ and

$$K'_i \leftarrow \mathsf{KMaul}(K''_i, R_\mathsf{A} \oplus R''_\mathsf{B}). \tag{1}$$

However, C_j obtains K'_i by computing $K'_i \leftarrow \mathsf{KMaul}(K_i, R''_\mathsf{A} \oplus R_\mathsf{B})$. Therefore, K'_i relates to 6 variants: $K_i, c''_i, R_\mathsf{A}, R''_\mathsf{A}, R_\mathsf{B}$ and R''_B.

We first analyze how K_i and c''_i are generated: C_j generates $(c_i, K_i) \leftarrow\!\!\$\ \mathsf{KEM.Encap}(pk_i)$. Then F_j computes $c'_i \leftarrow \mathsf{Rer}(\mathsf{CMaul}(c_i, \bar{r}_2^{-1}), \bar{\gamma}_i)$. After that, F_i computes $c''_i \leftarrow \mathsf{Rer}(\mathsf{CMaul}(c'_i, r_2^{-1}), \gamma_i)$. By the ciphertext malleability of KEM, the underlying session key is

$$K''_i = \mathsf{KMaul}(K_i, r_2^{-1} \oplus \bar{r}_2^{-1}). \tag{2}$$

Second, we analyze how R_B and R''_B are generated: C_j picks $R_\mathsf{B} \leftarrow\!\!\$\ \mathcal{R}_{cm}$. Then F_j computes $R'_\mathsf{B} \leftarrow R_\mathsf{B} \oplus \bar{r}'_0$, and F_i computes $R''_\mathsf{B} \leftarrow R'_\mathsf{B} \oplus r'_0$. Therefore, we have

$$R''_\mathsf{B} = R_\mathsf{B} \oplus \bar{r}'_0 \oplus r'_0. \tag{3}$$

Now, combining (1), (2) and (3), we have that K'_i computed by C_i is $\mathsf{KMaul}(K''_i, R_\mathsf{A} \oplus R_\mathsf{B} \oplus \bar{r}'_0 \oplus r'_0) = \mathsf{KMaul}(K_i, R_\mathsf{A} \oplus R_\mathsf{B} \oplus \bar{r}'_0 \oplus r'_0 \oplus r_2^{-1} \oplus \bar{r}_2^{-1})$.

Third, we analyze how R_A and R''_A are generated: C_i picks $R_\mathsf{A} \leftarrow\!\!\$\ \mathcal{R}_{cm}$. Then F_i computes $\overline{C}' \leftarrow \mathsf{Rerand}'(\mathsf{MaulCom}'(\overline{C}, r'_0 \oplus r_2^{-1}), r'_1)$. The underlying message is $R'_\mathsf{A} = R_\mathsf{A} \oplus r'_0 \oplus r_2^{-1}$. F_j computes $\overline{C}'' \leftarrow \mathsf{Rerand}'(\mathsf{MaulCom}'(\overline{C}', \bar{r}'_0 \oplus \bar{r}_2^{-1}), \bar{r}'_1)$. The underlying message is $R''_\mathsf{A} = R'_\mathsf{A} \oplus \bar{r}'_0 \oplus \bar{r}_2^{-1}$. Therefore, we have

$$R''_\mathsf{A} = R_\mathsf{A} \oplus r'_0 \oplus r_2^{-1} \oplus \bar{r}'_0 \oplus \bar{r}_2^{-1}. \tag{4}$$

C_j computes $K'_i = \mathsf{KMaul}(K_i, R''_\mathsf{A} \oplus R_\mathsf{B}) = \mathsf{KMaul}(K_i, R_\mathsf{A} \oplus R_\mathsf{B} \oplus r'_0 \oplus r_2^{-1} \oplus \bar{r}'_0 \oplus \bar{r}_2^{-1})$, which is the same with what C_i computes.

Computing K'_j. Upon receiving $v'' := (\tilde{c}'', c''_i, R''_\mathsf{B})$ from F_i, C_i computes $K'_j = \mathsf{KMaul}(K_j, R_\mathsf{A} \oplus R''_\mathsf{B}) = \mathsf{KMaul}(K_j, R_\mathsf{A} \oplus R_\mathsf{B} \oplus \bar{r}'_0 \oplus r'_0)$. C_j computes $K''_j = \mathsf{Decap}(sk_2, c''_j)$ and

$$K'_j = \mathsf{KMaul}(K''_j, R''_\mathsf{A} \oplus R_\mathsf{B}). \tag{5}$$

Protocol srAKE

The first message: Upon receiving input (UsrSession, $sid, ssid, \mathsf{S}$), U generates the first message as follows:

- C_i generates $(\widetilde{pk}, \widetilde{sk}) \leftarrow\!\$\ \mathsf{bKEM.Gen}(\mathsf{pp})$, commits to $\widetilde{pk}$ by computing $C \leftarrow \mathsf{Com}(\widetilde{pk}, \rho, r)$ where $r \leftarrow\!\$\ \mathcal{R}_{com}$, picks $R_{\mathsf{A}} \leftarrow\!\$\ \mathcal{R}_{cm}$ and computes $\overline{C} \leftarrow \mathsf{Com}'(R_{\mathsf{A}}, \rho', r_s)$ where $r_s \leftarrow\!\$\ \mathcal{R}'_{com}$. Then C_i computes $(c_j, K_j) \leftarrow\!\$\ \mathsf{KEM.Encap}(pk_j)$ and sends $u \coloneqq (C, c_j, \overline{C})$ to F_i.
- F_i computes $C' \leftarrow \mathsf{Rerand}(\mathsf{MaulCom}(C, r_0), r_1)$ where $(r_0, r_1) \leftarrow\!\$\ (\mathcal{R}_{mcom}, \mathcal{R}_{com})$, and $c'_j \leftarrow \mathsf{Rer}(\mathsf{CMaul}(c_j, r_2), \gamma_j)$, $\overline{C}' \leftarrow \mathsf{Rerand}'(\mathsf{MaulCom}'(\overline{C}, r'_0 \oplus r_2^{-1}), r'_1)$ where $(r'_0, r_2) \leftarrow\!\$\ \mathcal{R}^2_{cm}, r'_1 \leftarrow\!\$\ \mathcal{R}_{com}, \gamma_j \leftarrow\!\$\ \mathcal{R}_{re}$, sends $u' \coloneqq (C', c'_j, \overline{C}')$ to S.

The second message: Upon receiving $u' \coloneqq (C', c'_j, \overline{C}')$ from U, S generates the second message as follows:

\\ F_j sanitizes u' into u'' in the same manner as F_i.

- F_j computes $C'' \leftarrow \mathsf{Rerand}(\mathsf{MaulCom}(C', \bar{r}_0), \bar{r}_1)$ where $(\bar{r}_0, \bar{r}_1) \leftarrow\!\$\ (\mathcal{R}_{mcom}, \mathcal{R}_{com})$, and $c''_j \leftarrow \mathsf{Rer}(\mathsf{CMaul}(c'_j, \bar{r}_2), \bar{\gamma}_j)$, $\overline{C}'' \leftarrow \mathsf{Rerand}'(\mathsf{MaulCom}'(\overline{C}', \bar{r}'_0 \oplus \bar{r}_2^{-1}), \bar{r}'_1)$ where $(\bar{r}'_0, \bar{r}_2) \leftarrow\!\$\ \mathcal{R}^2_{cm}, \bar{r}'_1 \leftarrow\!\$\ \mathcal{R}_{com}, \bar{\gamma}_j \leftarrow\!\$\ \mathcal{R}_{re}$, sends $u'' \coloneqq (C'', c''_j, \overline{C}'')$ to C_j.
- Upon receiving $u'' \coloneqq (C'', c''_j, \overline{C}'')$ from F_j and input (SrvSession, $sid, ssid, \mathsf{U}$), C_j picks $R_{\mathsf{B}} \leftarrow\!\$\ \mathcal{R}_{cm}$, computes $(c_i, K_i) \leftarrow\!\$\ \mathsf{KEM.Encap}(pk_i)$, $\tilde{c} \leftarrow \mathsf{bKEM.Cg}(\tilde{r})$ where $\tilde{r} \leftarrow\!\$\ \mathsf{bKEM.Rg}(\mathsf{pp})$, and sends $v \coloneqq (\tilde{c}, c_i, R_{\mathsf{B}})$ to F_j.
- Upon receiving $v \coloneqq (\tilde{c}, c_i, R_{\mathsf{B}})$, F_j computes $\tilde{c}' \leftarrow \mathsf{CKeyMaul}(\tilde{c}, \bar{r}_0)$, $c'_i \leftarrow \mathsf{Rer}(\mathsf{CMaul}(c_i, \bar{r}_2^{-1}), \bar{\gamma}_i)$ where $\bar{\gamma}_i \leftarrow\!\$\ \mathcal{R}_{re}$, $R'_{\mathsf{B}} \leftarrow R_{\mathsf{B}} \oplus \bar{r}'_0$, and sends $v' \coloneqq (\tilde{c}', c'_i, R'_{\mathsf{B}})$ to U.

The third message: Upon receiving $v' \coloneqq (\tilde{c}', c'_i, R'_{\mathsf{B}})$ from S, U generates the third message as follows:

- F_i computes $\tilde{c}'' \leftarrow \mathsf{CKeyMaul}(\tilde{c}', r_0)$, $c''_i \leftarrow \mathsf{Rer}(\mathsf{CMaul}(c'_i, r_2^{-1}), \gamma_i)$ where $\gamma_i \leftarrow\!\$\ \mathcal{R}_{re}$, $R''_{\mathsf{B}} \leftarrow R'_{\mathsf{B}} \oplus r'_0$, returns $v'' \coloneqq (\tilde{c}'', c''_i, R''_{\mathsf{B}})$ to C_i. \\ F_i behaves identically to F_j.
- Upon receiving v'' from F_i, C_i computes $x \leftarrow \mathsf{Open}(\rho, C, r)$, $\overline{x} \leftarrow \mathsf{Open}(\rho', \overline{C}, r_s)$, and sends $(x, \overline{x})$ to F_i.
- F_i computes $\widetilde{pk} \leftarrow \mathsf{Verif}(\rho, C, x)$, $R_{\mathsf{A}} \leftarrow \mathsf{Verif}(\rho', \overline{C}, \overline{x})$, $x' \leftarrow \mathsf{OpenRerand}(\mathsf{xMaul}(x, r_0), r_1)$ and $\overline{x}' \leftarrow \mathsf{OpenRerand}(\mathsf{xMaul}(\overline{x}, r'_0 \oplus r_2^{-1}), r'_1)$. If $\widetilde{pk} \neq \perp \wedge R_{\mathsf{A}} \neq \perp$, F_i sends $(x', \overline{x}')$ to S; else it drops $(x, \overline{x})$.
- Upon receiving $(x', \overline{x}')$ from U, F_j computes $\widetilde{pk}' \leftarrow \mathsf{Verif}(\rho, C', x')$, $R'_{\mathsf{A}} \leftarrow \mathsf{Verif}(\rho', \overline{C}', \overline{x}')$, $x'' \leftarrow \mathsf{OpenRerand}(\mathsf{xMaul}(x', \bar{r}_0), \bar{r}_1)$ and $\overline{x}'' \leftarrow \mathsf{OpenRerand}(\mathsf{xMaul}(\overline{x}', \bar{r}'_0 \oplus \bar{r}_2^{-1}), \bar{r}'_1)$. If $\widetilde{pk}' \neq \perp \wedge R'_{\mathsf{A}} \neq \perp$, F_j sends $(x'', \overline{x}'')$ to C_j; otherwise, it drops the message. \\ F_j behaves identically to F_i.

Computation of the session key:

- C_i computes three keys as follows: $\widetilde{K} \leftarrow \mathsf{bKEM.Decap}(\widetilde{sk}, \tilde{c}'')$, $K''_i \leftarrow \mathsf{KEM.Decap}(sk_i, c''_i)$, $K'_i \leftarrow \mathsf{KMaul}(K''_i, R_{\mathsf{A}} \oplus R''_{\mathsf{B}})$, $K'_j \leftarrow \mathsf{KMaul}(K_j, R_{\mathsf{A}} \oplus R''_{\mathsf{B}})$. Finally, C_i computes $K \leftarrow H(\mathsf{ctxt} \| K'_i \| K'_j \| \widetilde{K})$.
- C_j computes $\widetilde{pk}'' \leftarrow \mathsf{Verif}(\rho, C'', x'')$, $R''_{\mathsf{A}} \leftarrow \mathsf{Verif}(\rho', \overline{C}'', \overline{x}'')$. If $\widetilde{pk}'' = \perp \vee R''_{\mathsf{A}} = \perp$, C_j drops the message; else it computes $\widetilde{K} \leftarrow \mathsf{bKEM.Kg}(\widetilde{pk}'', \tilde{r})$, $K'_i \leftarrow \mathsf{KMaul}(K_i, R''_{\mathsf{A}} \oplus R_{\mathsf{B}})$, $K''_j \leftarrow \mathsf{KEM.Decap}(sk_j, c''_j)$, $K'_j \leftarrow \mathsf{KMaul}(K''_j, R''_{\mathsf{A}} \oplus R_{\mathsf{B}})$. Finally, C_j computes $K \leftarrow H(\mathsf{ctxt} \| K'_i \| K'_j \| \widetilde{K})$.

Figure 12: Our srAKE protocol. Let $\mathsf{ctxt} = sid \| ssid \| \mathsf{aux}$ where $\mathsf{aux} \coloneqq \mathsf{C}_i \| \mathsf{C}_j \| pk_i \| pk_j$.

We analyze how c_j'' is generated. First, C_i generates $(c_j, K_j) \leftarrow\!\$\ \mathsf{KEM.Encap}(pk_j)$. Then F_i computes $c_j' \leftarrow \mathsf{Rer}(\mathsf{CMaul}(c_j, r_2), \gamma_j)$. After that, F_j computes $c_j'' \leftarrow \mathsf{Rer}(\mathsf{CMaul}(c_j', \bar{r}_2), \bar{\gamma}_j)$. By the ciphertext malleability of KEM, we have

$$K_j'' = \mathsf{KMaul}(K_j, r_2 \oplus \bar{r}_2). \tag{6}$$

Combining (4), (5) and (6), we have that K_j' computed by C_j is equal to $\mathsf{KMaul}(K_j, R_\mathsf{A} \oplus R_\mathsf{B} \oplus r_0' \oplus r_2^{-1} \oplus \bar{r}_0' \oplus \bar{r}_2^{-1} \oplus r_2 \oplus \bar{r}_2) = \mathsf{KMaul}(K_j, R_\mathsf{A} \oplus R_\mathsf{B} \oplus r_0' \oplus \bar{r}_0')$.

Computing $\widetilde{K}$. C_i obtains $\widetilde{K}$ by computing $\mathsf{bKEM.Decap}(\widetilde{sk}, \tilde{c}'')$. $\tilde{c}''$ is obtained from $\tilde{c}'' \leftarrow \mathsf{CKeyMaul}(\tilde{c}', r_0)$ and

$$\tilde{c}' \leftarrow \mathsf{CKeyMaul}(\tilde{c}, \bar{r}_0). \tag{7}$$

Note that $\widetilde{pk}' = \mathsf{bKEM.KeyMaul}(\widetilde{pk}, r_0)$. By the key malleability of bKEM, we have

$$\mathsf{bKEM.Decap}(\widetilde{sk}, \tilde{c}'') = \mathsf{bKEM.Decap}(\widetilde{sk}', \tilde{c}'). \tag{8}$$

Moreover, we have $\widetilde{pk}'' = \mathsf{bKEM.KeyMaul}(\widetilde{pk}', \bar{r}_0)$. Combining with (7), it holds that

$$\mathsf{bKEM.Decap}(\widetilde{sk}'', \tilde{c}) = \mathsf{bKEM.Decap}(\widetilde{sk}', \tilde{c}'). \tag{9}$$

C_j computes $\widetilde{K} \leftarrow \mathsf{bKEM.Kg}(\widetilde{pk}'', \tilde{r})$ where $\tilde{c} = \mathsf{bKEM.Cg}(\tilde{r})$. By the blindness of bKEM, we have

$$\mathsf{bKEM.Kg}(\widetilde{pk}'', \tilde{r}) = \mathsf{bKEM.Decap}(\widetilde{sk}'', \tilde{c}). \tag{10}$$

Combining (8), (9) and (10), we have that both C_i and C_j compute the same $\widetilde{K}$.

D.3 Proof of Lemma 4

Proof. The behaviors of $\mathcal{S}$ are shown as follows. Initially, $\mathcal{S}$ creates a list L_H for recording the queries and answers to the RO H. Upon receiving an RO query $H(m)$, if m is new, $\mathcal{S}$ chooses a fresh random value $h \leftarrow\!\$\ \{0,1\}^w$, returns h and stores (m, h) into L_H; else $\mathcal{S}$ retrieves (m, h') from L_H and returns h'. Moreover, $\mathcal{S}$ generates $(\rho, ek) \leftarrow\!\$\ \mathsf{KGen}(1^\lambda)$, $(\rho', ek') \leftarrow\!\$\ \mathsf{KGen}'(1^\lambda)$, $\mathsf{pp} \leftarrow \mathsf{bKEM.Setup}(1^\lambda)$, $\mathsf{pp}' \leftarrow \mathsf{KEM.Setup}(1^\lambda)$ and picks $(pk_i, sk_i) \leftarrow\!\$\ \mathsf{KEM.Gen}(\mathsf{pp}')$, $(pk_j, sk_j) \leftarrow\!\$\ \mathsf{KEM.Gen}(\mathsf{pp}')$ as long-term key pairs for U and S, respectively. $\mathcal{S}$ shares the public keys pk_i, pk_j and the public parameters ρ, ρ',pp and pp' with $\mathcal{A}$. Here, the extraction keys ek and ek' are written explicitly for simulation.

Initiator corruption. When U is corrupted, $\mathcal{S}$ simulates the behaviors of S for U. The session key in the ideal world is a uniformly random value sampled by $\mathcal{F}_{\mathsf{AKE}}$. $\mathcal{S}$ is required to ensure consistency between the session key output by $\mathcal{F}_{\mathsf{AKE}}$ and the session key computed by $\mathcal{A}$:

- $\mathcal{S}$ reveals sk_i to $\mathcal{A}$.
- Upon receiving $u' = (C', c_j', \overline{C}')$ from $\mathcal{A}$, $\mathcal{S}$ computes $\widetilde{pk}' \leftarrow \mathsf{Extract}(ek, C')$, $R_\mathsf{A}' \leftarrow \mathsf{Extract}(ek', \overline{C}')$, simulates C_j and F_j to generate $v' = (\tilde{c}', c_i', R_\mathsf{B}')$ and compute $K_i^{**}, K_j^{**}, \widetilde{K}^{**}$. Then $\mathcal{S}$ picks $\kappa^* \leftarrow\!\$\ \{0,1\}^w$ and sends $(\textsc{NewKey}, sid, ssid, \mathsf{U}, \kappa^*)$ to $\mathcal{F}_{\mathsf{AKE}}$.
- Upon receiving κ which is sent from $\mathcal{F}_{\mathsf{AKE}}$ to (corrupted) U, $\mathcal{S}$ sends v' to $\mathcal{A}$. Now $\mathcal{A}$ is able to compute the session key by querying $H(sid \| ssid \| \mathsf{aux} \| K_i^{**} \| K_j^{**} \| \widetilde{K}^{**})$. If $\mathcal{A}$ makes such an RO query, then $\mathcal{S}$ returns κ to $\mathcal{A}$ and records $(sid \| ssid \| \mathsf{aux} \| K_i^{**} \| K_j^{**} \| \widetilde{K}^{**}, \kappa)$ into L_H.
- Upon receiving $(x', \overline{x}')$ from $\mathcal{A}$, $\mathcal{S}$ sends $(\textsc{NewKey}, sid, ssid, \mathsf{S}, \kappa)$ to $\mathcal{F}_{\mathsf{AKE}}$.

Note that in this corruption case, the indistinguishability of the simulation from the real-world execution relies on the extractability of Π_{Com} and Π'_{Com}, which enables $\mathcal{S}$ to detect the RO queries related to session keys and program the RO correctly.

Responder corruption. When the responder S is corrupted, $\mathcal{S}$ behaves as follows:

- $\mathcal{S}$ reveals sk_j to $\mathcal{A}$.
- $\mathcal{S}$ simulates C_i and F_i to generate $u' := (C', c'_j, \overline{C}')$ and sends u' to $\mathcal{A}$.
- Upon receiving $v' := (\tilde{c}', c'_i, R'_{\mathsf{B}})$ from $\mathcal{A}$, $\mathcal{S}$ computes $x, \overline{x}, x', \overline{x}', K_i^*, K_j^*$ and $\widetilde{K}^*$ according to the protocol specification.
- $\mathcal{S}$ picks $\kappa^* \leftarrow\!\!\$\ \{0,1\}^w$ and sends $(\textsc{NewKey}, sid, ssid, \mathsf{S}, \kappa^*)$ to $\mathcal{F}_{\mathsf{AKE}}$.
- Upon receiving κ sent from $\mathcal{F}_{\mathsf{AKE}}$ to (corrupted) S, $\mathcal{S}$ checks the list L_H:
  - If $\exists\ (sid\|ssid\|\mathsf{aux}\|K_i^*\|K_j^*\|\widetilde{K}^*, \cdot) \in L_H$, then abort.
  - Else, add $(sid\|ssid\|\mathsf{aux}\|K_i^*\|K_j^*\|\widetilde{K}^*, \kappa)$ to L_H.
- Send $(\textsc{NewKey}, sid, ssid, \mathsf{U}, \kappa)$ to $\mathcal{F}_{\mathsf{AKE}}$.
- Send the opening message $(x', \overline{x}')$ to $\mathcal{A}$.

This simulation is perfect unless $\mathcal{S}$ aborts. Let F denote the event that $\mathcal{S}$ aborts. Observe that F happens if and only if $\mathcal{A}$ queries $H(sid\|ssid\|\mathsf{aux}\|K_i^*\|K_j^*\|\widetilde{K}^*)$ before receiving $(x', \overline{x}')$ from $\mathcal{S}$. We show that if $\mathcal{A}$ makes such a query before receiving $(x', \overline{x}')$, then we can build an adversary $\mathcal{B}_h$ to break the computationally hiding property of Π_{Com}.

$\mathcal{B}_h$ simulates the protocol execution in the same way as $\mathcal{S}$ except for the generation of commitment $\tilde{c}'$. Specifically, after generating $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{bKEM.Setup}(1^\lambda)$ in the protocol simulation, $\mathcal{B}_h$ plays a hiding experiment against Π_{Com} defined in Fig. 11: Upon receiving the public parameter ρ of Π_{Com}, $\mathcal{B}_h$ picks $(\widetilde{pk}_0, \widetilde{sk}_0) \leftarrow\!\!\$\ \mathsf{bKEM.Gen}(\mathsf{pp})$, $(\widetilde{pk}_1, \widetilde{sk}_1) \leftarrow\!\!\$\ \mathsf{bKEM.Gen}(\mathsf{pp})$ and outputs $(\widetilde{pk}_0, \widetilde{pk}_1)$ in the computationally hiding experiment. Upon receiving a commitment c, $\mathcal{B}_h$ sets $C' = c$ and continues the simulation of srAKE. Upon receiving $v' := (\tilde{c}', c'_i, \overline{C}')$ from $\mathcal{A}$, $\mathcal{B}_h$ retrieves a key $\widetilde{K} = \widetilde{K}^*$ from $(sid\|ssid\|\mathsf{aux}\|K_i^*\|K_j^*\|\widetilde{K}^*, \cdot) \in L_H$ such that $\widetilde{K} = \mathsf{bKEM.Decap}(\widetilde{sk}_0, \tilde{c}')$ or $\widetilde{K} = \mathsf{bKEM.Decap}(\widetilde{sk}_1, \tilde{c}')$. Since we assume that the event F happens, $\mathcal{B}_h$ can always find such a query and a key $\widetilde{K}$. If $\widetilde{K} = \mathsf{bKEM.Decap}(\widetilde{sk}_0, \tilde{c}')$, then $\mathcal{B}_h$ returns $b' = 0$; otherwise, it returns $b' = 1$. It holds that $\Pr[b = b'] = 1$, which implies that $\mathcal{B}_h$ breaks the computationally hiding property of Π_{Com}. Therefore, if Π_{Com} is computationally hiding, we have $\Pr[F] \le \mathsf{negl}(\lambda)$, which completes the proof of Lemma 4.

□

D.4 Proof of Lemma 5

Proof. We start with the real-world protocol execution of srAKE, and by sequential modifications, we end up with the ideal-world execution of $\mathcal{S}$ and $\mathsf{Wrap}(\mathcal{F}_{\mathsf{AKE}})$. We show that the modification in each game is undetectable by $\mathcal{Z}$. Let $\mathtt{Hyb}_i$ denote the i-th hybrid game, and G_i denote the event that $\mathcal{Z}$ outputs 1 in the end of $\mathtt{Hyb}_i$. Let Real and Ideal denote the binary output of $\mathcal{Z}$ in the real world and ideal world, respectively. In this proof, we say a session $\langle ssid, \mathcal{P}, \mathcal{P}'\rangle$ is *actively attacked* if any of the three keys input to H in this session differs from that in session $\langle ssid, \mathcal{P}', \mathcal{P}\rangle$; otherwise it is treated as *passively observed*. A session $\langle ssid, \mathcal{P}, \mathcal{P}'\rangle$ is called COMPROMISED if the adversary $\mathcal{A}$ actively attacks this session and $\mathcal{P}'$ is COMPROMISED, and a session $\langle ssid, \mathcal{P}, \mathcal{P}'\rangle$ is called INTERFERED if the adversary $\mathcal{A}$ actively attacks this session but $\mathcal{P}'$ is not COMPROMISED. The sessions executed by S are called S sessions, while those executed by U are called U sessions. Let q_{U} be the number of U sessions, i.e., the number of USRSESSION queries sent from $\mathcal{Z}$ to all user U identities. Also, let q_{S} be the number of S sessions. Let $(sid\|ssid\|\mathsf{aux}\|K_i^*\|K_j^*\|\widetilde{K}^*)$ and $(sid\|ssid\|\mathsf{aux}\|K_i^{**}\|K_j^{**}\|\widetilde{K}^{**})$ be inputs to H when computing the keys of sessions $\langle ssid, \mathsf{U}, \mathsf{S}\rangle$ and $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$, respectively.

$\mathtt{Hyb}_0$ (executing real world protocol) : This is the real-world execution of srAKE where $\mathcal{A}$ controls the whole network and interacts with parties. Let $(u' = (C', c'_j, \overline{C}'), v' = (\tilde{c}', c'_i, R'_{\mathsf{B}}), (x', \overline{x}'))$ be the messages sent by honest parties, which are generated according to the protocol specification. Let $u'' = (C'', c''_j, \overline{C}''), v'' = (\tilde{c}'', c''_i, R''_{\mathsf{B}})$ and $(x'', \overline{x}'')$ be the corresponding messages received from $\mathcal{A}$. Let $\widetilde{pk}'$ and $\widetilde{pk}''$ be the underlying messages of C' and C'', respectively. It holds that

$$\Pr[G_0] = \Pr[\mathsf{Real} = 1].$$

$\mathtt{Hyb}_1$ (introducing a simulator): In this game, the protocol execution is moved into a single machine (i.e., simulator) denoted as $\mathcal{S}$. $\mathcal{S}$ simulates the execution of srAKE and answers RO queries to H by maintaining a hash list L_H, as described in the initial part of the proof of Lemma 4. $\mathcal{S}$ also generates $(\rho, ek) \leftarrow\!\$\ \mathsf{KGen}(1^\lambda), (\rho', ek') \leftarrow\!\$\ \mathsf{KGen}'(1^\lambda)$. Moreover, upon receiving a query $(\textsc{UsrSession}, sid, ssid, \mathsf{U}, \mathsf{S})$, $\mathcal{S}$ creates a session record $\langle ssid, \mathsf{U}, \mathsf{S}, \perp\rangle$ if $ssid$ was not used before by (simulated) U. When computing the session key κ, U updates the record to $\langle ssid, \mathsf{U}, \mathsf{S}, \kappa\rangle$. Symmetrically, the session record $\langle ssid, \mathsf{S}, \mathsf{U}, \perp\rangle$ is introduced and updated on S-side. Compared to $\mathtt{Hyb}_0$, $\mathtt{Hyb}_1$ only makes syntactical changes. Therefore, we have

$$\Pr[G_1] = \Pr[G_0].$$

$\mathtt{Hyb}_2$ (modifying the key computation method) : This game is a preparation for the following hybrid games. Upon receiving u'' from $\mathcal{A}$, $\mathcal{S}$ generates v' as specified by srAKE, then extracts $\widetilde{pk}''$ and R''_{A} from u'' using ek and ek', and subsequently computes $K_i^{**}, K_j^{**}, \widetilde{K}^{**}$.

When the simulated U calculates the session key κ, $\mathcal{S}$ checks if there exists a tuple $(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3, h) \in L_H$ such that $(K_1, K_2, K_3) = (K_i^*, K_j^*, \widetilde{K}^*)$. If such a tuple exists, $\mathcal{S}$ sets $\kappa \leftarrow h$. Otherwise, $\mathcal{S}$ sets $\kappa \leftarrow\!\$\ \{0,1\}^w$, but does not record κ into L_H until $\mathcal{A}$ makes the corresponding RO query.

During the simulation of S computing the session key κ, if session $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$ is passively observed, then set the key to be that of the corresponding U-session record $\langle ssid, \mathsf{U}, \mathsf{S}, \cdot\rangle$. Else $\mathcal{S}$ verifies if there exists a tuple $(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3, h) \in L_H$ such that $(K_1, K_2, K_3) = (K_i^{**}, K_j^{**}, \widetilde{K}^{**})$. If found, $\mathcal{S}$ sets $\kappa = h$. Otherwise, $\mathcal{S}$ selects $\kappa \leftarrow\!\$\ \{0,1\}^w$ but does not record κ into L_H until $\mathcal{A}$ makes the corresponding RO query. The only difference between $\mathtt{Hyb}_1$ and $\mathtt{Hyb}_2$ is the time to record RO queries, which is undetectable by $\mathcal{Z}$. It holds that

$$\Pr[G_2] = \Pr[G_1].$$

$\mathtt{Hyb}_3$ (randomizing key computation of U in COMPROMISED sessions) : The only difference of this game from $\mathtt{Hyb}_2$ is that upon key computation of a COMPROMISED session $\langle ssid, \mathsf{U}, \mathsf{S}\rangle$, $\mathcal{S}$ picks $\kappa \leftarrow\!\$\ \{0,1\}^w$ as the session key and does not check the list L_H. κ is only recorded into list L_H when $\mathcal{A}$ makes an RO query $(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3)$ where $(K_1, K_2, K_3) = (K_i^*, K_j^*, \widetilde{K}^*)$. $\mathcal{Z}$ cannot distinguish $\mathtt{Hyb}_3$ from $\mathtt{Hyb}_2$ except when it makes such an RO query before receiving $(x', \overline{x}')$ from simulated U. This case is close to the case where U is honest and S is corrupt. As is argued in the S-corruption case in Appendix D.3, if $\mathcal{A}$ makes such an RO query, then we can build another adversary to break the computationally hiding property of Π_{Com}. The corresponding reduction is identical to that in Appendix D.3 and is omitted. We have

$$|\Pr[G_3] - \Pr[G_2]| \leq \mathsf{negl}(\lambda).$$

$\mathtt{Hyb}_4$ (randomizing key computation of U in INTERFERED sessions) : This game is the same as $\mathtt{Hyb}_3$ except for setting the key of session $\langle ssid, \mathsf{U}, \mathsf{S}\rangle$ to be random if it is INTERFERED, i.e., $\mathcal{A}$ actively attacked this session but S is not COMPROMISED, and the corresponding hash value is not programmed until S is COMPROMISED.

$\mathcal{Z}$ cannot distinguish $\mathtt{Hyb}_4$ from $\mathtt{Hyb}_3$ unless it makes an RO query $(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3)$ such that $(K_1, K_2, K_3) = (K_i^*, K_j^*, \widetilde{K}^*)$, before S is COMPROMISED. Let Q denote the event that $\mathcal{Z}$ makes such an RO query before S is COMPROMISED. Then we have

$$|\Pr[G_4] - \Pr[G_3]| \leq \Pr[Q].$$

To bound the probability of Q, we define another event T, which occurs when $\mathcal{Z}$ makes an RO query $H(sid\|ssid\|\mathsf{aux}\| \cdot \| \cdot \|K_j^*)$ before S is COMPROMISED. We have

$$\Pr[Q] \leq \Pr[T].$$

Now we argue that, if T happens, then we can build an adversary $\mathcal{D}$ that breaks the OW-PCA security game against KEM using $\mathcal{Z}$ as a subroutine with non-negligible probability: $\mathcal{D}$ plays the OW-PCA game against its challenger. At the same time, $\mathcal{D}$ interacts with $\mathcal{Z}$ as in $\mathtt{Hyb}_3$, with the following modifications: Initially, upon receiving (pk, c), $\mathcal{D}$ picks $k \leftarrow\!\!\$\ [q_{\mathsf{U}}]$, sets the public key of S to be $pk_j := pk$, and $c'_j := c$ for k-th session $(ssid_k, \mathsf{U}, \mathsf{S})$ of U. Assume that S generates the randomness $R'_{\mathsf{B}k}$ and receives $(C''_k, c''_{jk}, \overline{C}''_k)$ in session $\langle ssid_k, \mathsf{S}, \mathsf{U}\rangle$.

Since $\mathcal{D}$ lacks the knowledge of S's private key sk_j, all computations and queries involving sk_j should be handled with care. Specifically, $\mathcal{D}$ is required to guarantee the consistency between the computation of session keys and queries to RO. $\mathcal{D}$ needs to identify whether it holds that $K_3 = \mathsf{KEM.Decap}(sk_j, c''_{j\delta}) \oplus R''_{\mathsf{A}\delta} \oplus R'_{\mathsf{B}\delta}$ in an RO query of the form $(sid\|ssid_\delta\|\mathsf{aux}\|K_1\|K_2\|K_3)$ which is the input to RO when computing the key of session $\langle ssid_\delta, \mathsf{U}, \mathsf{S}\rangle$ or $\langle ssid_\delta, \mathsf{S}, \mathsf{U}\rangle$, where $ssid_\delta$ refers to some session between U and S, $R'_{\mathsf{B}\delta}$ is sent by S, $c''_{j\delta}$ is received by S, and $R''_{\mathsf{A}\delta}$ is computed by S (by extracting $C''_{\mathsf{A}\delta}$, i.e., the commitment to $R''_{\mathsf{A}\delta}$) in session $\langle ssid_\delta, \mathsf{S}, \mathsf{U}\rangle$. The only way is to make a PCO query $(c''_{j\delta}, K_2 \oplus R''^{-1}_{\mathsf{A}\delta} \oplus R'^{-1}_{\mathsf{B}\delta})$. If the output is 1, then $K_2 \oplus R''^{-1}_{\mathsf{A}\delta} \oplus R'^{-1}_{\mathsf{B}\delta}$ is a valid decapsulation of $c''_{j\delta}$. If it holds that $K_2^* \oplus R''^{-1}_{\mathsf{A}k} \oplus R'^{-1}_{\mathsf{B}k} = \mathsf{KEM.Decap}(sk_j, c'_j)$ in an RO query of the form $(sid\|ssid_k\|\mathsf{aux}\| \cdot \|K_2^*\|\cdot)$, $\mathcal{D}$ immediately returns $K_2^* \oplus R''^{-1}_{\mathsf{A}k} \oplus R'^{-1}_{\mathsf{B}k}$ and wins the OW-PCA security game against KEM. We have

$$\Pr[T] \leq q_{\mathsf{U}} \cdot \mathrm{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{KEM}}(\mathcal{D}, \lambda).$$

Therefore, it holds that

$$|\Pr[G_4] - \Pr[G_3]| \leq \Pr[Q] \leq q_{\mathsf{U}} \cdot \mathrm{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{KEM}}(\mathcal{D}, \lambda).$$

$\mathtt{Hyb}_5$(randomizing key computation of U in passively observed sessions): This game is the same as $\mathtt{Hyb}_4$, except that the session key of U is set to be $\kappa \leftarrow\!\!\$\ \{0,1\}^w$ without maintaining the consistency of RO queries if the session is passively observed. In this case, the session key of S in the session $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$ is set to κ. $\mathcal{Z}$ cannot distinguish $\mathtt{Hyb}_5$ and $\mathtt{Hyb}_4$ unless it makes an RO query $H(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3)$ such that $(K_1, K_2, K_3) = (K_i^*, K_j^*, \widetilde{K}^*)$. The event that $\mathcal{Z}$ makes such an RO query is denoted as Q'. If Q' happens, we can construct an adversary $\mathcal{D}'$ that breaks the strong one-wayness of bKEM with non-negligible probability.

Let $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda), (pk, sk) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{pp})$ and $r \leftarrow\!\!\$\ \mathsf{Rg}(\mathsf{pp}), c \leftarrow \mathsf{Cg}(r)$. Given (pk, c), $\mathcal{D}'$ aims to return (K, pk', c') such that $K = \mathsf{bKEM.Decap}(sk, c') = \mathsf{bKEM.Kg}(pk', r)$ where $pk' \in \mathcal{PK}_{\mathsf{bKEM}}$ and $c' \in \mathcal{C}_{\mathsf{bKEM}}$ (the ciphertext space of bKEM). $\mathcal{D}'$ interacts with $\mathcal{A}$ as in $\mathtt{Hyb}_4$ except that it picks $k \leftarrow\!\!\$\ [q_{\mathsf{U}}]$, sets $\widetilde{pk}' := pk$ and $\tilde{c}' := c$ in k-th session $(ssid_k, \mathsf{U}, \mathsf{S})$, i.e., $\mathcal{D}'$ guesses that $\mathcal{Z}$ would make an RO query on $(sid\|ssid_k\|\mathsf{aux}\| \cdot \| \cdot \|K_3)$ such that $K_3 = \mathsf{bKEM.Decap}(\widetilde{sk}', \tilde{c}'') = \mathsf{bKEM.Decap}(\widetilde{sk}'', \tilde{c}')$, where $\tilde{c}''$ is the bKEM ciphertext received by U in session $(ssid_k, \mathsf{U}, \mathsf{S})$, $\widetilde{sk}'$ and $\widetilde{sk}''$ represent the underlying private keys corresponding to $\widetilde{pk}'$ and $\widetilde{pk}''$ (i.e., the public key extracted by S in session $(ssid_k, \mathsf{S}, \mathsf{U})$), respectively. If $\mathcal{Z}$ makes such

a query, $\mathcal{D}$ returns $(K_3, \widetilde{pk}'', c'')$ to its challenger and breaks the strong one-wayness of bKEM. Let $\mathrm{Adv}^{\mathsf{stOW}}_{\mathsf{bKEM}}(\mathcal{D}', \lambda)$ denote the probability that $\mathcal{D}'$ breaks the strong one-wayness of bKEM. We have $\Pr[Q'] \leq q_{\mathsf{U}} \cdot \mathrm{Adv}^{\mathsf{stOW}}_{\mathsf{bKEM}}(\mathcal{D}', \lambda)$. Therefore, it holds that

$$|\Pr[G_5] - \Pr[G_4]| \leq q_{\mathsf{U}} \cdot \mathrm{Adv}^{\mathsf{stOW}}_{\mathsf{bKEM}}(\mathcal{D}', \lambda).$$

$\mathtt{Hyb}_6$ (randomizing key computation of S in INTERFERED sessions): This game is the same as $\mathtt{Hyb}_5$, except that the session key of an INTERFERED S session $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$ is set to be uniformly random without maintaining the consistency of the RO. The corresponding hash value is not programmed until U is COMPROMISED.

$\mathcal{Z}$ cannot distinguish $\mathtt{Hyb}_6$ from $\mathtt{Hyb}_5$ unless it makes an RO query $H(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3)$ before U is COMPROMISED such that $(K_1, K_2, K_3) = (K_i^{**}, K_j^{**}, \widetilde{K}^{**})$. Let Q'' denote the event that $\mathcal{Z}$ makes such an RO query before U is COMPROMISED. Then we have

$$|\Pr[G_6] - \Pr[G_5]| \leq \Pr[Q''].$$

To bound the probability of Q'', we define another event M, which occurs when $\mathcal{Z}$ makes an RO query of the form $H(sid\|ssid\|\mathsf{aux}\|K_1\| \cdot \|\cdot)$ such that $K_1 = K_i^{**}$ before U is COMPROMISED. We have

$$\Pr[Q''] \leq \Pr[M].$$

We show that if M happens, then we can construct an adversary $\mathcal{D}''$ that breaks the OW-PCA security of KEM using $\mathcal{Z}$ as a subroutine. $\mathcal{D}''$ plays the OW-PCA security game against KEM. $\mathcal{D}''$ also interacts with $\mathcal{Z}$ as in $\mathtt{Hyb}_5$, with the following modifications: Given (pk, c) in the OW-PCA game, $\mathcal{D}''$ picks $k \leftarrow\!\!\$\ [q_{\mathsf{S}}]$, sets the public key of U to be $pk_i := pk$, and $c'_i := c$ for k-th session $(ssid_k, \mathsf{S}, \mathsf{U})$ of S. Assume that U generates the randomness $R'_{\mathsf{A}k}$ and receives $(c''_{ik}, R''_{\mathsf{B}k})$ in session $\langle ssid_k, \mathsf{U}, \mathsf{S}\rangle$.

Since $\mathcal{D}''$ has no knowledge of U's private key sk_i, all computations and queries related to sk_i should be carefully handled. Similar to that in $\mathtt{Hyb}_4$, $\mathcal{D}$ is required to guarantee the consistency between the computation of session keys and queries to RO. $\mathcal{D}$ needs to identify whether it holds that $K_1 = \mathsf{KEM.Decap}(sk_i, c''_{i\delta}) \oplus R'_{\mathsf{A}\delta} \oplus R''_{\mathsf{B}\delta}$ in an RO query of the form $(sid\|ssid_\delta\|\mathsf{aux}\|K_1\|K_2\|K_3)$ which is the input to RO when computing the key of session $\langle ssid_\delta, \mathsf{S}, \mathsf{U}\rangle$ or $\langle ssid_\delta, \mathsf{U}, \mathsf{S}\rangle$, where $ssid_\delta$ refers to some session between U and S, $R'_{\mathsf{A}\delta}$ is generated by U and $(c''_{i\delta}, R''_{\mathsf{B}\delta})$ is received by U in session $\langle ssid_\delta, \mathsf{U}, \mathsf{S}\rangle$. The only way is to make a PCO query $(c''_{i\delta}, K_1 \oplus R'^{-1}_{\mathsf{A}\delta} \oplus R''^{-1}_{\mathsf{B}\delta})$ to the challenger in OW-PCA security game. If the output is 1, then K_2 is a valid decapsulation of $c''_{i\delta}$. If it holds that $K_1^* \oplus R'^{-1}_{\mathsf{A}k} \oplus R''^{-1}_{\mathsf{B}k} = \mathsf{KEM.Decap}(sk_i, c'_i)$ in an RO query of the form $(sid\|ssid_k\|\mathsf{aux}\|K_1^*\| \cdot \|\cdot)$, $\mathcal{D}$ immediately returns $K_1^* \oplus R'^{-1}_{\mathsf{A}k} \oplus R''^{-1}_{\mathsf{B}k}$ and wins the OW-PCA security game against KEM. We have

$$\Pr[M] \leq q_{\mathsf{S}} \cdot \mathrm{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{KEM}}(\mathcal{D}'', \lambda).$$

Therefore, it holds that

$$|\Pr[G_6] - \Pr[G_5]| \leq \Pr[Q''] \leq q_{\mathsf{S}} \cdot \mathrm{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{KEM}}(\mathcal{D}'', \lambda).$$

$\mathtt{Hyb}_7$ (advancing key computation of COMPROMISED S sessions): This game is the same as $\mathtt{Hyb}_6$, except that upon $\mathcal{S}$ sending v' to $\mathcal{A}$ in session $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$, if $\mathcal{A}$ makes an RO query $(sid\|ssid\|\mathsf{aux}\|K_1\|K_2\|K_3)$ such that (1) $(K_1, K_2, K_3) = (K_i^{**}, K_j^{**}, \widetilde{K}^{**})$, (2) U is COMPROMISED and (3) $\widetilde{pk}'' \neq \widetilde{pk}' \vee K''_j \neq K'_j \vee R''_{\mathsf{A}} \neq R'_{\mathsf{A}}$, then U is impersonated and session $\langle ssid, \mathsf{S}, \mathsf{U}\rangle$ is COMPROMISED, $\mathcal{S}$ responds with $\kappa \leftarrow\!\!\$\ \{0,1\}^w$ and updates the record

$\langle ssid, \mathsf{S}, \mathsf{U}, \bot \rangle$ to $\langle ssid, \mathsf{S}, \mathsf{U}, \kappa \rangle$ in advance. This modification is to maintain consistency between the session key computation and RO queries of $\mathcal{A}$ when $u'' \neq u'$ is not generated as it would be by a firewall and $\mathcal{A}$ makes a correct RO query related to the session key: The third condition implies that the adversary either generates u'' itself or mauls u' like a firewall.

In the former case, upon receiving v' from simulated S, $\mathcal{A}$ is able to make a correct RO query to compute the session key since U is COMPROMISED. Therefore, $\mathcal{S}$ updates the record $\langle ssid, \mathsf{S}, \mathsf{U}, \bot \rangle$ in advance. If $\mathcal{A}$ makes such an RO query, then $\mathcal{S}$ successfully detects it and programs the output. In the latter case, the adversary behaves like a firewall and has no knowledge of the underlying $\widetilde{pk}'$ of C' and R'_{A} of $\overline{C}'$ due to the computationally hiding property of Π_{Com} and Π'_{Com}. Therefore, even though the adversary can maul the commitments of $\widetilde{pk}'$ and R'_{A}, it cannot compute K_i^{**}, K_j^{**} or $\widetilde{K}^{**}$ with non-negligible probability. We have

$$|\Pr[G_7] - \Pr[G_6]| \leq \mathsf{negl}(\lambda).$$

$\mathtt{Hyb}_8$ (introducing the functionality $\mathcal{F}$): This game is the same as $\mathtt{Hyb}_7$, except that we introduce an ideal functionality $\mathcal{F}$. The interfaces of USRSESSION, SRVSESSION, COMPROMISE, IMPERSONATE and NEWKEY queries are also incorporated into $\mathcal{F}$, as in $\mathcal{F}_{\mathsf{AKE}}$. Upon key compromise or interfering behaviors from $\mathcal{A}$, $\mathcal{S}$ sends the corresponding COMPROMISE or IMPERSONATE messages to $\mathcal{F}_{\mathsf{AKE}}$ as in Fig. 10. There is no change in the way of computing session keys between $\mathtt{Hyb}_8$ and $\mathtt{Hyb}_7$. We have

$$\Pr[G_8] = \Pr[G_7].$$

It is easy to see that this game is exactly the ideal process, i.e., $\mathcal{F} = \mathcal{F}_{\mathsf{AKE}}$. Therefore, we have

$$\Pr[G_8] = \Pr[\mathsf{Ideal}].$$

Finally, we conclude that in case where the cores and firewalls of both parties are honest, the probability of $\mathcal{Z}$ distinguishing the real-world protocol execution from the ideal-world process is negligible based on the OW-PCA security of KEM and strong one-wayness of bKEM, extractability and computationally hiding property of Π_{Com} and Π'_{Com}.

The lemma follows immediately. □

E Omitted Instantiations

E.1 Overview of Instantiations

The instantiations of C_i and F_i are depicted in Fig. 13. The public parameters are $\mathsf{pp} \leftarrow \mathsf{bKEM.Setup}(1^\lambda)$, $\mathsf{pp}' \leftarrow \mathsf{KEM.Setup}(1^\lambda)$, $\rho \leftarrow\!\$\ \mathsf{KGen}(1^\lambda), \rho' \leftarrow\!\$\ \mathsf{KGen}'(1^\lambda)$, in which pp includes $g \in \mathbb{G}$, ρ includes (g_x, h_x) and ρ' includes (g_y, h_y) s.t. $h_x = g_x^{t_x}, h_y = g_y^{t_y}$ where $g_x, g_y \in \mathbb{G}, (t_x, t_y) \leftarrow\!\$\ \mathbb{Z}_p^{*2}$. The public/private key pairs of C_i and C_j are (pk_i, sk_i) and (pk_j, sk_j), respectively, which are generated by $\mathsf{KEM.Gen}(\mathsf{pp})$. The instantiations of C_j and F_j are depicted in Fig. 14, where F_j shares the same code with F_i. For a pair of values $X_r = (A, B)$, let $X_{r,1} \coloneqq A, X_{r,2} \coloneqq B$.

E.2 Security Analysis of Instantiations

E.2.1 The Security of Π_{Com} and Π'_{Com}

Theorem 2. *Π_{Com} is rerandomizable, extractable, tight, malleable with respect to bKEM, and computationally hiding if DDH assumption holds.*

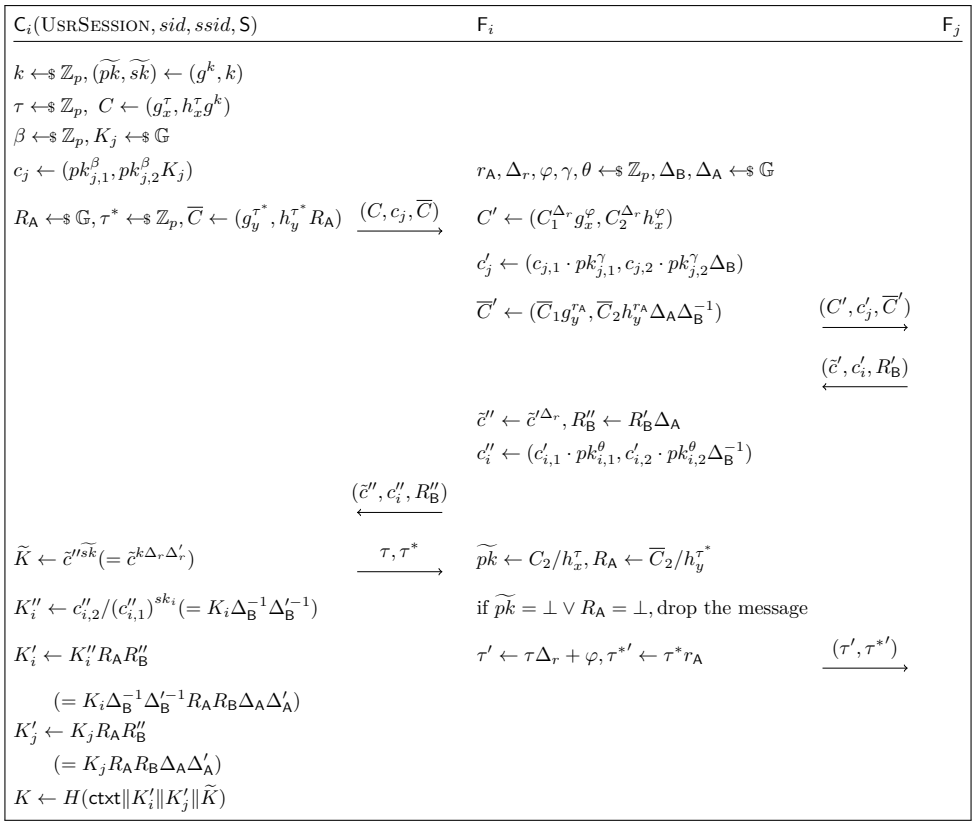


Figure 13: Instantiations of C_i and F_i, where $\mathsf{ctxt} = sid \| ssid \| \mathsf{aux}$, Δ'_A and Δ'_B are generated by F_j in Fig. 14. Let $C = (C_1, C_2), \overline{C} = (\overline{C}_1, \overline{C}_2)$.

Proof. We prove the properties of Π_{Com} as follows.

Rerandomizability. For any commitment $C = (C_1, C_2)$ and $r \leftarrow\$ \mathbb{Z}_p$, $(C'_1, C'_2) = \mathsf{Rerand}(C, r) = (C_1 \cdot g^r, C_2 \cdot h^r)$ is uniformly random. For an opening x, let $x' = \mathsf{OpenRerand}(x, r) = x + r$, $C_1 = g^x$. We have $C'_1 = g^{x'}$, $\mathsf{Verif}(\rho, C, x) = C_2 / h^x, \mathsf{Verif}(\rho, C', x') = C'_2 / h^{x'} = C_2 / h^x$ and thus we have $\mathsf{Verif}(\rho, C', x') = \mathsf{Verif}(\rho, C, x)$. Otherwise, if $C_1 \neq g^x$, then we have $C'_1 \neq g^{x'}$, $\mathsf{Verif}(\rho, C', x') = \mathsf{Verif}(\rho, C, x) = \perp$. Therefore, Π_{Com} is rerandomizable.

Extractability. Let $(\rho, ek) \leftarrow\$ \mathsf{KGen}(1^\lambda)$, where $\rho := (\mathbb{G}, \mathbb{Z}_p, \mathbb{G}^2, g, h)$ and $h = g^{ek}$ as described in the specification of Π_{Com}. For any $m \in \mathbb{G}$ and $r \in \mathbb{Z}_p$, let $C = \mathsf{Com}(m, \rho; r) = (g^r, h^r \cdot m)$, we have $\mathsf{Verif}(\rho, C, r) = (h^r \cdot m) / h^r = m$ and $\mathsf{Extract}(ek, C) = (h^r \cdot m) / (g^r)^{ek} = m$. Therefore, it holds that $\Pr[\mathsf{Verif}(\rho, C, r) = m \wedge \mathsf{Extract}(ek, C) \neq m] \leq \mathsf{negl}(\lambda)$.

Tightness. In Π_{Com}, for any ciphertext C and message m, there exists only one opening y such that $C = (g^y, h^y \cdot m)$. Therefore, Π_{Com} is a tight commitment.

Malleability with respect to bKEM. First, we show that Π_{Com} is malleable: For any commitment $C := (C_1, C_2)$, its opening x and randomness $r \leftarrow\$ \mathcal{R}_{com}$, let $m = \mathsf{Verif}(\rho, C, x) = C_2 / h^x, m' = \mathsf{mMaul}(m, r) = m^r$. We compute $C' = \mathsf{MaulCom}(C, r) = (C_1^r, C_2^r)$ and $x' = \mathsf{xMaul}(x, r) = x \cdot r$. Then we have $\mathsf{Verif}(\rho, C', x') = C_2^r / h^{x'} = (C_2 / h^x)^r$. Therefore, it holds that $\mathsf{Verif}(\rho, C', x') = m'$. Since r is uniformly picked from $\mathcal{R}_{com}$, the distribution of m' is uniformly random. Let $\mathsf{pp} \leftarrow\$ \mathsf{bKEM.Setup}(1^\lambda), (pk, sk) \leftarrow\$ \mathsf{bKEM.Gen}(\mathsf{pp})$. For any $r \in \mathcal{R}_{com}$, we have $\mathsf{bKEM.KeyMaul}(pk; r) = \mathsf{mMaul}(pk, r) = pk^r$, which means that Π_{Com} is malleable with respect to bKEM.

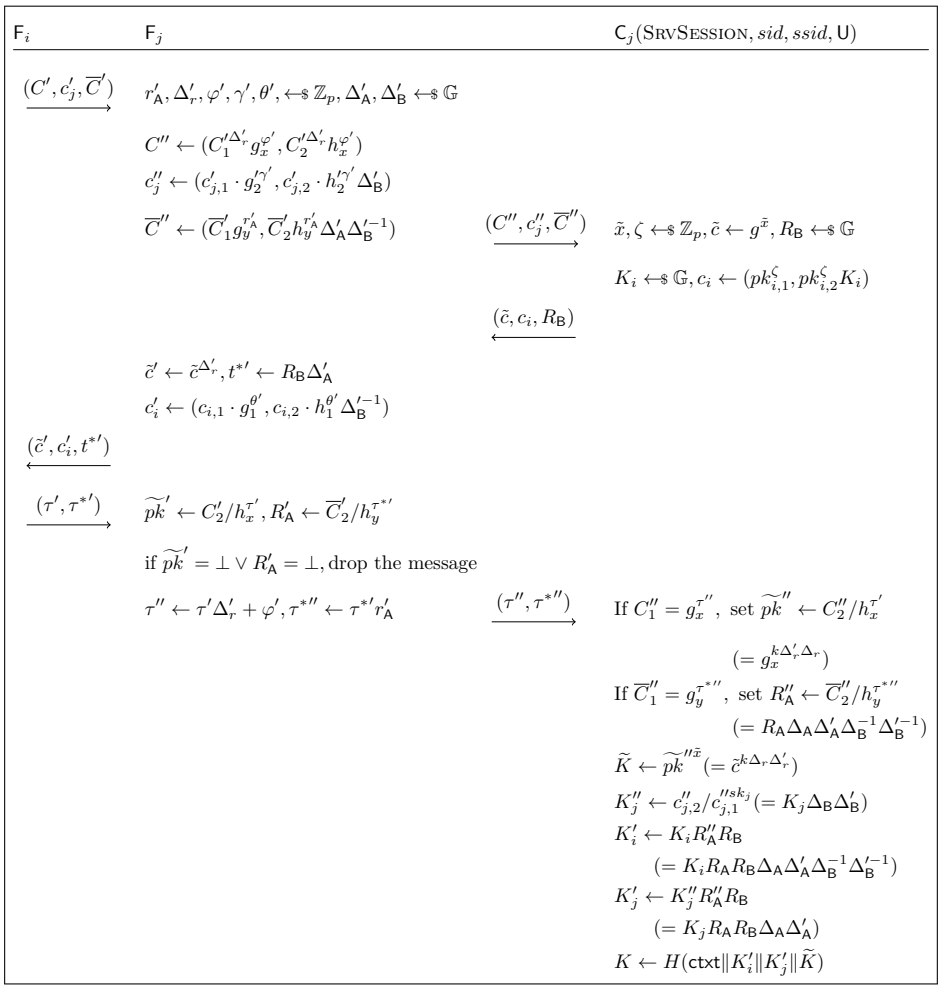


Figure 14: Instantiations of C_j and F_j.

Computational hiding. Let $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ be a PPT adversary that plays hiding experiment against a challenger $\mathcal{C}_\Pi$ that runs Π_{Com}. We prove the computational hiding property of Π_{Com} via hybrid arguments. Define Game 0 to be the security game played between $\mathcal{A}$ and $\mathcal{C}_\Pi$ in the hiding experiment. Let W_i denote the event that the experiment outputs 1 in Game i.

Game 0. This game is the initial hiding experiment. Specifically, $\mathcal{C}_\Pi$ first runs $\mathsf{KGen}(1^\lambda)$ and gets $\rho := (\mathbb{G}, \mathbb{Z}_p, \mathbb{G}^2, g, h)$. Upon receiving two plaintexts m_0 and m_1, the challenger picks $b \leftarrow\!\!\$\ \{0,1\}, y \leftarrow\!\!\$\ \mathbb{Z}_p$, computes $C \leftarrow (g^y, h^y \cdot m_b)$ and returns C to $\mathcal{A}$. Upon receiving the guessing bit b' from $\mathcal{A}$, $\mathcal{C}_\Pi$ sets the output of this game to 1 if $b = b'$; otherwise, it is set to 0. Let $\Pr[W_0] = \frac{1}{2} + \epsilon$ where $0 \le \epsilon \le 1/2$.

Game 1. This game is the same as Game 0, except that $\mathcal{C}_\Pi$ picks $y \leftarrow\!\!\$\ \mathbb{Z}_p, Z^* \leftarrow\!\!\$\ \mathbb{G}$ and returns $C = (g^y, Z^* \cdot m_b)$ instead of $C = (g^y, h^y \cdot m_b)$ to $\mathcal{A}$. Now we show that if $\mathcal{A}$ can distinguish Game 0 from Game 1, then we can build an adversary $\mathcal{B}$ that breaks the DDH assumption (Definition 16) using $\mathcal{A}$ as a subroutine. Initially, $\mathcal{B}$ receives a DDH challenge (g, X, Y, Z). Upon receiving two plaintexts m_0 and m_1 from $\mathcal{A}$, $\mathcal{B}$ picks $b \leftarrow\!\!\$\ \{0,1\}$ and computes $C \leftarrow (Y, Z \cdot m_b)$ and returns C to $\mathcal{A}$. When $\mathcal{A}$ outputs a guessing bit b', $\mathcal{B}$ will output 1 if $b = b'$; otherwise, it will output 0. Observe that if (X, Y, Z) is a DH-triple, then $\mathcal{B}$ is simulating Game 0 for $\mathcal{A}$; else $\mathcal{B}$ is simulating Game 1 for $\mathcal{A}$. Therefore, we have $|\Pr[W_0] - \Pr[W_1]| \le \mathrm{DDHAdv}[\mathcal{B}, \mathbb{G}](\lambda)$.

Observe that in Game 1, the ciphertext C is uniformly random. We have $\Pr[W_1] = \frac{1}{2}$. Therefore, it holds that $\mathrm{DDHAdv}[\mathcal{A}, \mathbb{G}](\lambda) \ge \epsilon$. If DDH assumption holds for $\mathbb{G}$, then the probability that the hiding experiment outputs 1 is $\frac{1}{2} + \mathsf{negl}(\lambda)$. □

Theorem 3. *Π'_{Com} is rerandomizable, extractable, tight, malleable with respect to KEM, and computationally hiding if the DDH assumption holds.*

Proof. The security proof of Π'_{Com} is identical to that of Π_{Com}, except for the malleability w.r.t. KEM. Let $m' = \mathsf{mMaul}'(m, r) = m \cdot r$. Since $\mathsf{KMaul}(K, r) = K \cdot r$, we have $\mathsf{KMaul}(K, m') = \mathsf{KMaul}(K, m \cdot r)$. Therefore, Π'_{Com} is malleable with respect to KEM. □

E.2.2 The Security of DHK

Theorem 4. *DHK satisfies strong one-wayness if the iCDH assumption holds.*

Proof. We show that if $\mathcal{A}$ breaks the strong one-wayness of DHK, we can construct another adversary $\mathcal{B}$ that breaks the iCDH assumption.

Initially, $\mathcal{B}$ is given an iCDH problem instance (g, g^a, g^b). Initially, $\mathcal{B}$ sets g^a as the public key of DHK and g^b as the challenge ciphertext. Then $\mathcal{B}$ sends $(pk, c) := (g^a, g^b)$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs (K^*, pk^*, c^*). $\mathcal{B}$ returns (K^*, pk^*, c^*) as a solution to the iCDH problem instance. If $\mathcal{A}$ breaks the strong one-wayness of bKEM, it holds that $K^* = \mathsf{Decap}(sk^*, c) = \mathsf{Decap}(sk, c^*)$, i.e., $\mathtt{DH}(pk^*, g^b, K^*) = \mathtt{DH}(g^a, c^*, K^*) = 1$, and thus $\mathcal{B}$ breaks the iCDH assumption. Obviously, we have $\mathrm{Adv}^{\mathsf{stOW}}_{\mathsf{DHK}}(\mathcal{A}, \lambda) \le \mathrm{Adv}^{\mathsf{iCDH}}_{\mathbb{G}}(\mathcal{B}, \lambda)$. □

E.2.3 The Security of ElGamal-KEM

Theorem 5. *The ElGamal-KEM scheme satisfies OW-PCA if the SDH assumption holds.*

Proof. Consider the OW-PCA security game (defined in Fig. 5) played between a challenger $\mathcal{C}$ running the $\mathsf{ElGamal\text{-}KEM}$ scheme and an adversary $\mathcal{A}$. Let W denote the event that the game outputs 1. By the definition of OW-PCA, we have $\mathrm{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{ElGamal\text{-}KEM}}(\mathcal{A}, \lambda) := \Pr[W]$. We show that if W occurs, we can construct another adversary $\mathcal{B}$ that breaks the SDH assumption.

Initially, $\mathcal{B}$ is given an SDH problem instance (g, g^a, g^b). All the simulations of the OW-PCA game follow the game specification except the following modifications: In the starting phase

of the OW-PCA game, $\mathcal{B}$ picks $c_2 \leftarrow\!\!\$\ \mathbb{G}$, sets $pk \leftarrow g^a$ as the public key of $\mathsf{ElGamal\text{-}KEM}$ and $c \coloneqq (c_1 = g^b, c_2)$ as the challenge ciphertext. Then $\mathcal{B}$ sends (pk, c) to $\mathcal{A}$ in the OW-PCA game. If $\mathcal{A}$ makes a PCO query on $(c' = (c'_1, c'_2), K')$, then $\mathcal{B}$ queries its *restricted* DDH oracle $(c'_1, c'_2/K')$ and forwards the response to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a key K^*. $\mathcal{B}$ returns K^* as a solution to the SDH problem instance. If W happens, then it holds that $K^* = c_2/g^{ab}$, and thus $\mathcal{B}$ breaks the SDH assumption. Obviously, we have $\mathrm{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{ElGamal\text{-}KEM}}(\mathcal{A}, \lambda) \leq \mathrm{Adv}^{\mathsf{SDH}}_{\mathbb{G}}(\mathcal{B}, \lambda)$. $\square$