Offline Assisted Group Key Exchange

Colin Boyd Gareth T. Davies Kristian Gjøsteen Yao Jiang

NTNU, Norwegian University of Science and Technology, Trondheim, Norway
{colin.boyd,gareth.davies,kristian.gjosteen,yao.jiang}@ntnu.no

February 2, 2018

Abstract

We design a group key exchange protocol where most of the participants remain offline until they wish to compute the key. This is well suited to a cloud storage environment where users are often offline, but have online access to the server which can assist in key exchange. We define and instantiate a new primitive, a blinded KEM, which we show can be used in a natural way as part of our generic protocol construction. Our new protocol has a security proof based on a well-known model for group key exchange. Our protocol provides a restricted form of forward secrecy which we argue is as strong as can be achieved in practice. Our protocol is efficient, requiring Diffie–Hellman with a handful of standard public key operations per user in our concrete instantiation.

Keywords: Authenticated Key Exchange, Group Key Exchange, Forward Secrecy, Cloud Storage, Blinded Key Encapsulation

1 Introduction

Consider the scenario of a collaborative project. Isabel the initiator wants to store some files encrypted on a cloud server so that her collaborators Robin and Rolf can access them later. Isabel somehow needs to share the key she used to encrypt with Robin and Rolf. Of course if they were all online at the same time, the three of them could use group key exchange (GKE) to acquire a shared key. In many practical scenarios this is not the case, so her options are more limited. She could use standard hybrid encryption: pick a symmetric key k, encrypt that under the public keys of each of the recipients, and encryptions of the files under the key k in the cloud. However this means that if at any point in the future any recipient's long-term secret key is compromised then confidentiality of the initial files is lost. Isabel wants some kind of forward secrecy. Since Isabel will store the encrypted files on the cloud server anyway, it makes sense to ask if it is possible to have the server assist her in the key exchange with her collaborators?

We present a key exchange protocol, named OAGK, that solves the above concerns. Briefly, Isabel first performs half of a key exchange with the server, which holds some secret value. Then a responder who comes online later need only perform the other half of the protocol with the server. When it comes to the details, more is needed to protect against an active adversary. We provide an abstract protocol version making use of a novel primitive, which we call a *blinded KEM*, and a concrete instantiation using Diffie–Hellman.

1.1 Secure sharing in cloud storage

It is today commonplace for users, both individuals and companies, to store important and valuable data in the cloud. Some of the well-known advantages compared with using local storage are: flexibility of storage, automatic backup facilities, and access from anywhere. Another potential advantage is ease of data sharing. While many users are content to trust a cloud provider to secure their data, for many users such a convenient trust assumption is unreasonable or even impossible due to legal regulations. Therefore there are several cloud storage providers who support user-side encryption of data so that the storage provider never has access to the keys or data[1].

In this paper we will not assume that the cloud storage provider is (fully) trusted and therefore users encrypt their data before storage. This means that simple access control by the cloud provider is no longer sufficient to manage data sharing – instead there is a need to share the keys used to encrypt the data. There are today many group key exchange protocols [15, 4] which could potentially be used in such situations, but the problem with applying such protocols in the cloud scenario is that they require interaction so that no user can complete the protocol and accept the shared key until all users have participated. The alternative is then to use a key transport solution, where the originating user can send the key to other users. However, this solution fails to provide any form of forward secrecy; if any user's long-term decryption key is compromised at any time in the future then the shared key becomes vulnerable.

Our goal is then to provide a group key exchange protocol which does not require synchronous interaction between users but at the same time provides forward secrecy to the largest extent possible. At the same time we demand a simple and practical solution which is amenable to a formal security analysis.

Schemes to support secure sharing of cloud storage have been proposed several times in the literature [17, 18, 19, 7]. However, they do not fit our requirements for one reason or another. Note that in this context many papers are concerned with the formation of dynamic groups, where there is an unfortunate clash of terminology. For dynamic groups, *forward secrecy* is often used to mean the property that deleted group members cannot access new material, and *backward secrecy* means that added group members cannot access previously deleted material. In this paper we use forward secrecy in the sense traditional in (group) key exchange, namely that compromise of long-term keys does not compromise previously completed sessions.

- Mona [19] is a system to provide anonymous sharing of user data. It relies on a trusted third party which generates all user keys so it fails to achieve our trust requirement. The system also relies on pairings and lacks a formal security analysis.

- Tresorit [17, 18] applies a dynamic tree-based group Diffie–Hellman variant in order to share keys within a group. However, this solution does not provide forward secrecy since users store long-term keys which can be used to decrypt shared data.

- Chu et al. [7] apply key-aggregate encryption to allow users to be issued keys which can decrypt a chosen set of files. This is a flexible approach but they do not actually address the issue of distributing keys (mentioning only secure email) and the approach does not provide forward secrecy.

1.2 Forward secrecy without interaction

Forward secrecy is typically achieved through the use of interaction with Diffie–Hellman or other ephemeral keys. Using ephemeral keys for confidentiality and long-term keys only for authentication ensures that later release of long-term secrets does not reveal the session key. This presents an inherent conflict in our requirements to allow for offline users. Indeed, a simple generic argument implies that forward secrecy without interaction is impossible: without interaction the recipient cannot provide an ephemeral input and therefore the recipient's long-term key alone must be sufficient to recover the session key. There have been two recent proposals to work around this argument in different ways.

[1]This practice is confusingly often called *zero knowledge* in commercial circles.

- Marlinspike and Perrin [20] propose the "Extended Triple Diffie-Hellman" (X3DH) protocol which avoids the generic impossibility argument by adjusting what is meant by an *ephemeral* key. Instead of requiring each recipient to generate a new ephemeral key during the protocol, users publish some identifying information and some *pre-keys* to the server before the initiator begins communication with the server. These pre-keys are used as if they are ephemeral, but of course they are now less ephemeral than keys generated at the time of the protocol run. Compromise of the corresponding secret keys will compromise session keys generated using them. We note, in addition, that the X3DH protocol does not satisfy our requirements for other reasons since it is limited to a single recipient and lacks a formal security analysis.

- Green and Miers [12] introduced puncturable encryption which was applied by Günther et al. [13] to provide low-latency key exchange. With this approach the generic impossibility result is avoided by adjusting what is meant by a *long-term* key. Each time a message is received, the long-term key is updated (punctured) so that later long-term key compromise cannot be used to decrypt that same message. Thus the long-term key is no longer static but evolves over time. Forward secrecy with puncturable encryption relies crucially on the assumption that the protocol (single) message arrives at the receiver. Until that happens the receiver private key is not updated and so the encrypted data is vulnerable to receiver compromise. In the cloud scenario this is a significant problem since we assume that the receiver may be offline and may not be expecting any such message. In addition we note that current concrete implementations of puncturable encryption rely on less efficient pairing-based primitives and require increased storage and secure deletion properties at the receiver.

In our protocol we choose a simple approach by sharing some temporary values at the semi-trusted online server. In order to break our protocol an adversary must compromise both the server and one of the users. The server stores a medium-term key which is deleted after the protocol run is complete (or after a time-out) after which compromise of the server is allowed. Like the above two schemes, our protocol does not achieve forward secrecy in the strongest sense, but we believe that we reach a comparable level of security. In addition, our protocol is simple and relies only on standard assumptions. We note that it would not be difficult to enhance our protocol with *forward secure encryption* [6] if receiver compromise is deemed a likely risk.

1.3 Efficiency

There are different ways to measure the efficiency of group key exchange protocols, including the number of protocol messages, the number of rounds of parallel messages, and the (average) computation per user. There exist theoretically efficient examples [2, 3] but most practical protocols employ a generalisation of the Diffie–Hellman protocol. One such generalisation is the well-known scheme of Burmester and Desmedt [5] which requires 2 rounds of communication and 3 exponentiations per user in its unauthenticated version.

An example of a modern optimised protocol is that of Gao et al. [11] which adds signatures to all messages and requires users to verify the signature on broadcast messages from all other users. In comparison our requirements are relatively modest. We require 3 rounds but do not use broadcast messages at all. The protocol participants perform 5 public key operations each, consisting of signature generation/verification, public key encryption/decryption and key encapsulation/decapsulation. As mentioned, we cannot achieve full forward secrecy as in interactive protocols such as that of Gao et al. [11].

1.4 Contributions

We regard the following as the main contributions of this paper.

- We introduce definitions and constructions for a new cryptographic primitive, blinded KEMs, which may find other applications. We describe this primitive and provide two secure constructions in Section 4.
- We propose a novel practical group key exchange protocol suitable for use in cloud storage. Our protocol is described in Section 5.
- We include a formal security analysis of our protocol in a strong security model with trust assumptions suited to the cloud scenario. The proof is in a security model which is detailed in Section 3.

2 Preliminaries

For a set S, denote $x \overset{\$}{\leftarrow} S$ to mean choosing x uniformly at random from S. We write **return** $b' \overset{?}{=} b$ as shorthand for **if** $b' = b$ **then return** 1; **else return** 0, with an output of 1 indicating successful adversarial behavior.

2.1 Public-key encryption

A public-key encryption scheme $\mathsf{PKE} = (\mathsf{KG_{pke}}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ is defined as follows. $\mathsf{KG_{pke}}$ takes as input some security parameter(s), if any, and outputs a public encryption key pk and a secret decryption key sk. Enc takes a message m and produces a ciphertext c using pk: $c \leftarrow \mathsf{Enc}_{pk}(m)$. Dec decrypts a ciphertext c using sk to recover m or in the case of failure a symbol $\perp$: $m/\perp \leftarrow \mathsf{Dec}_{sk}(c)$. Correctness requires that $m \leftarrow \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m))$ for all $m \in \mathcal{M}$.

We denote the usual advantage of an adaptive chosen ciphertext adversary $\mathcal{A}$ against real-or-random security for the public-key encryption scheme by $\mathbf{Adv}^{\mathsf{ror\text{-}cca2}}_{\mathsf{PKE}}(\mathcal{A})$. In our protocol's security proof, it is actually convenient to use a generalization of this notion, which we discuss in Appendix A.

2.2 Digital signatures

A signature scheme $\mathsf{DS} = (\mathsf{KG_{sig}}, \mathsf{Sign}, \mathsf{Verify})$ with message space $\mathcal{M}$ is defined as follows. $\mathsf{KG_{sig}}$ takes as input some security parameter(s), if any, and outputs a signing key sk and a public verification key vk. Sign creates a signature σ on a message m: $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$. Verify verifies that the signature on the message is in fact valid: $0/1 \leftarrow \mathsf{Verify}_{vk}(m, \sigma)$, with 1 indicating successful verification. Correctness requires that $\mathsf{Verify}_{vk}(m, \mathsf{Sign}_{sk}(m)) = 1$ for all $m \in \mathcal{M}$.

Definition 1. Let $\mathsf{DS} = (\mathsf{KG_{sig}}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme. Then the $\mathsf{suf\text{-}cma}$ advantage of an adversary $\mathcal{A}$ against DS is defined as

$$\mathbf{Adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{DS}}(\mathcal{A}) = \mathbf{Pr}[\mathbf{Exp}^{\mathsf{suf\text{-}cma}}_{\mathsf{DS}}(\mathcal{A}) = 1].$$

where the experiment $\mathbf{Exp}^{\mathsf{suf\text{-}cma}}_{\mathsf{DS}}(\mathcal{A})$ is given in Fig. 1.

Note that in the existential unforgeability under chosen message attack ($\mathsf{euf\text{-}cma}$) game the list $\mathsf{S_{LIST}}$ only keeps track of the messages sent by the adversary, so $\mathcal{A}$ is not allowed to output (m, σ_2) if she sent m to $\mathcal{O}.\mathsf{Sign}$ and received σ_1.

2.3 Hardness assumptions

Definition 2. Fix a cyclic group $\mathbb{G}$ of prime order q with generator g. The advantage of an algorithm $\mathcal{A}$ solving the *Decision Diffie-Hellman (DDH)* problem for $\mathbb{G}$ and g is

$$\mathbf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{A}) = \left|\Pr[\mathbf{Exp}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{A}) = 1] - \frac{1}{2}\right|$$

where the experiment $\mathbf{Exp}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{A})$ is given in Fig. 2.

```
$\mathbf{Exp}^{\mathsf{suf\text{-}cma}}_{\mathsf{DS}}(\mathcal{A})$ :
    $\mathsf{S_{LIST}} \leftarrow \emptyset$
    $sk, vk \leftarrow \mathsf{KG_{sig}}$
    $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}.\mathsf{Sign}}(vk)$
    if $\mathsf{Verify}_{vk}(m, \sigma)$ and $(m, \sigma) \notin \mathsf{S_{LIST}}$
        return $1$
    else
        return $0$
```

```
$\mathcal{O}.\mathsf{Sign}(m)$ :
    if $m \notin \mathcal{M}$ then
        return $\perp$
    $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$
    $\mathsf{S_{LIST}} \leftarrow \mathsf{S_{LIST}} \cup (m, \sigma)$
    return $\sigma$
```

Figure 1: The experiment defining suf-cma security for signature schemes.

```
$\mathbf{Exp}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{A})$ :
    $b \xleftarrow{\$} \{0, 1\}$
    $x, y, z \xleftarrow{\$} \mathbb{Z}_q$
    if $b = 1$
        $c \leftarrow g^{xy}$
    else
        $c \leftarrow g^{z}$
    $b' \leftarrow \mathcal{A}(g^x, g^y, c)$
    return $b' \overset{?}{=} b$
```

Figure 2: DDH experiment.

```
$\mathbf{Exp}^{\mathsf{CR}}_{\mathcal{F}}(\mathcal{A})$ :
    $\mathsf{f} \xleftarrow{\$} \mathcal{F}$
    $x, y \leftarrow \mathcal{A}()$
    if $x \neq y \wedge \mathsf{f}(x) = \mathsf{f}(y)$ then
        return $1$
    else
        return $0$
```

Figure 3: Collision resistance experiment.

Definition 3. Let $\mathcal{F}$ be a family of functions. The *collision resistance* advantage of an adversary $\mathcal{A}$ running in time t is

$$\mathbf{Adv}^{\mathsf{CR}}_{\mathcal{F}}(\mathcal{A}) = \left| \Pr[\mathbf{Exp}^{\mathsf{CR}}_{\mathcal{F}}(\mathcal{A}) = 1] \right|$$

where the experiment $\mathbf{Exp}^{\mathsf{CR}}_{\mathcal{F}}(\mathcal{A})$ is given in Fig. 3.

Note that in an abuse of notation, we sometimes write $\mathbf{Adv}^{\mathsf{CR}}_{\mathsf{f}}(\mathcal{A})$, with the understanding that the function family $\mathcal{F}$ exists and that the choice of a function f is done at some point.

3 GKE protocol model

The model described in this section is based on previous models for group key exchange such as those of Katz and Yung [15] and Bresson and Manulis [4]. This includes game-based security definitions.

3.1 Communication Model

A GKE protocol P is a collection of probabilistic algorithms that determines how oracles of the principals behave in response to signals (messages) from their environment.

Protocol participants and long-lived keys. Each principal V in the protocol is either a user U or a server S. In every session, each user may act as either an initiator I or a responder R. Each principal V holds long-term secret keys, and corresponding public keys of all principals are known to all.

Session identifiers and partner identifiers. Protocol principals maintain multiple instances, or sessions, that may be run simultaneously and we denote a session of principal V by the oracle $\prod^{\alpha}_{\mathsf{V}}$ with $\alpha \in \mathbb{N}$.

Each oracle $\prod_{\mathsf{V}}^{\alpha}$ is associated with the variables $\mathsf{status}_{\mathsf{V}}^{\alpha}$, $\mathsf{role}_{\mathsf{V}}^{\alpha}$, $pid_{\mathsf{V}}^{\alpha}$, $sid_{\mathsf{V}}^{\alpha}$, k_{V}^{α} as follows:

- $\mathsf{status}_{\mathsf{V}}^{\alpha}$ takes a value from $\{unused, ready, accepted, rejected\}$.
- $\mathsf{role}_{\mathsf{V}}^{\alpha}$ takes a value from: S, I, R.
- $pid_{\mathsf{V}}^{\alpha}$ contains a set of principals.
- $sid_{\mathsf{V}}^{\alpha}$ contains a string defined by the protocol.
- k_{V}^{α} the agreed session key (if any).

A session identifier, denoted *sid*, is a protocol-defined value stored at a principal intended to provide a link to other sessions in the same protocol run. A set of partner identifiers, denoted *pid*, contains the identities of all intended users in a session.

Each oracle $\prod_{\mathsf{V}}^{\alpha}$ is *unused* until initialization, by which it is told to act as a server or a user together with the long term secret keys. During initialization all oracles begin with $\mathsf{status}_{\mathsf{V}}^{\alpha} = ready$ and $\mathsf{role}_{\mathsf{V}}^{\alpha}$, $pid_{\mathsf{V}}^{\alpha}$, $sid_{\mathsf{V}}^{\alpha}$ and k_{V}^{α} all equal to $\perp$.

Executing the protocol. After the protocol starts, each oracle $\prod_{\mathsf{V}}^{\alpha}$ learns its partner identifier $pid_{\mathsf{V}}^{\alpha}$ and sends, receives and processes messages.

If the protocol at oracle $\prod_{\mathsf{V}}^{\alpha}$ *fails*, for example if signature verification or key confirmation fails, then the oracle changes its state to *rejected* and no longer responds to protocol messages. Otherwise, if V is a user, after computing k_{V}^{α} oracle $\prod_{\mathsf{V}}^{\alpha}$ changes its state to *accepted* and no longer responds to protocol messages, and if V is the server, oracle $\prod_{\mathsf{V}}^{\alpha}$ accepts after all responder oracles get their messages or expiration.

3.2 Security Notions

Adversarial model. An efficient adversary $\mathcal{A}$ interacts with sessions by using the set of queries defined below. This models the ability of $\mathcal{A}$ to completely control the network, deciding which instances run and obtaining access to other useful information. The Test query can only be asked once by $\mathcal{A}$ and is only used to measure adversary's success; it does not correspond to any actual adversary's ability.

- $\mathsf{Execute}(\mathcal{S})$: Input a set of unused oracles $\mathcal{S}$ which execute an honest run of the protocol. The oracles compute what the protocol specifies and returns the output messages.
- $\mathsf{Send}(\prod_{\mathsf{V}}^{\alpha}, m)$: Sends message m to oracle $\prod_{\mathsf{V}}^{\alpha}$. The oracle computes what the protocol defines, and sends back the output message (if any), together with the status of $\prod_{\mathsf{V}}^{\alpha}$.
- $\mathsf{Corrupt}(\mathsf{V})$: Outputs principal V's long-term secret key.
- $\mathsf{Reveal}(\prod_{\mathsf{V}}^{\alpha})$: Outputs session key k_{V}^{α} if oracle $\prod_{\mathsf{V}}^{\alpha}$ has accepted and holds some session key k_{V}^{α}.
- $\mathsf{Test}(\prod_{\mathsf{V}}^{\alpha})$: If oracle $\prod_{\mathsf{V}}^{\alpha}$ has status *accepted*, holding a session key k_{V}^{α}, then a bit b is randomly chosen and this query outputs the session key k_{V}^{α} if $b = 1$, or a random string from the session key space if $b = 0$.

Partnering. A secure GKE protocol should ensure that the session key established in an oracle $\prod_{\mathsf{V}}^{\alpha}$ is independent of session keys established in other sessions, except for the partners of $\prod_{\mathsf{V}}^{\alpha}$. This is modeled by allowing the adversary to reveal any session key except the one in the Test session and its partners. Informally, partnering is defined in such a way that oracles who are supposed to agree on the shared session key are partners.

Definition 4. Two oracles $\prod_{\mathsf{V}}^{\alpha}$ and $\prod_{\mathsf{W}}^{\beta}$ are *partners* if $pid_{\mathsf{V}}^{\alpha} = pid_{\mathsf{W}}^{\beta}$ and $sid_{\mathsf{V}}^{\alpha} = sid_{\mathsf{W}}^{\beta}$.

Freshness. The notion of freshness models the conditions on the adversary's behaviour that are required to prevent trivial wins.

Definition 5. An oracle $\prod_{\mathsf{V}}^{\alpha}$ is *fresh* if neither this oracle nor any of its partnered oracles have been asked a Reveal query, and either

- no server player nor any player in $pid_{\mathsf{V}}^{\alpha}$ was corrupted before every partnered oracle reached status *accepted*; or
- no player in $pid_{\mathsf{V}}^{\alpha}$ is ever corrupted.

Security Game. Bringing together everything we have introduced so far, we can describe the game that allows us to measure the advantage of an adversary against a GKE protocol.

Definition 6. Let P be a GKE protocol. The game $\mathbf{Exp}_{\mathsf{P}}^{\mathsf{ake}}(\mathcal{A})$ consists of the following three phases:

- *Initialization.* Each principal V runs the key generation algorithm to generate long-term key pairs. The secret keys are only known to the principal, while public keys are revealed to every principal and the adversary.
- *Queries.* The adversary $\mathcal{A}$ is allowed to make $\mathsf{Execute}$, Send, Reveal, $\mathsf{Corrupt}$, and Test queries. During this phase, $\mathcal{A}$ is only allowed to ask only one Test query to a fresh oracle, which should remain fresh until the end of this phase.
- *Guessing.* $\mathcal{A}$ outputs its guess b'.

The output of the game is 1 if $b = b'$, otherwise 0.

The *advantage* of the adversary $\mathcal{A}$ against the ake-security of P is

$$\mathbf{Adv}_{\mathsf{P}}^{\mathsf{ake}}(\mathcal{A}) = 2\left|\Pr[\mathbf{Exp}_{\mathsf{P}}^{\mathsf{ake}}(\mathcal{A}) = 1] - 1/2\right|.$$

4 Blinded KEM

The concept of using public-key encryption to transport keys for use in symmetric encryption is by now well studied [8, 9, 10, 16, 1, 14]. This primitive is known as a *key encapsulation mechanism* (KEM) and is used in conjunction with a *data encapsulation mechanism* (DEM) that models some symmetric encryption scheme. This KEM-DEM framework is widely deployed in internet protocols, however – as we mentioned earlier – it does not provide any forward secrecy. The cloud scenario allows the initiator to store the encapsulated key and the DEM ciphertext in some repository for the recipient to later retreive, but we ask: can the (untrusted) cloud give us some notion of forward secrecy of the key that the initator wishes to transport?

We introduce a new primitive, which we name a *blinded KEM*, that has two additional algorithms than traditional KEMs: a blinding algorithm takes some encapsulation[2] and adds a blinding value, and an unblinding algorithm (that requires an unblinding key created by the blinding algorithms) removes this blinding value from the blinded key. Note that this construction *does not generalize existing KEMs* since our decapsulation procedure works on blinded encapsulations rather than encapsulations.

The point of this new idealized primitive is to allow parties to safely *outsource* decapsulation by creating a blinded encapsulation, having someone else decapsulate and then unblinding the result. With careful key management, this idea will give us forward secrecy in our cloud scenario. We will develop this idea into a group key exchange protocol in the next section.

After providing a definition of this primitive's algorithms, we give two natural constructions (based on DH and RSA).

[2] We abuse nomenclature throughout the rest of the paper and use 'encapsulation' to refer to a key encapsulation that is yet to be blinded.

Definition 7. A *blinded key encapsulation mechanism (blinded KEM)* BKEM consists of five algorithms $(\mathsf{KG}_{\mathsf{BKEM}}, \mathsf{Encap}, \mathsf{Blind}, \mathsf{Decap}, \mathsf{Unblind})$. The *key generation* algorithm $\mathsf{KG}_{\mathsf{BKEM}}$ outputs an encapsulation key ek and a decapsulation key dk. The *encapsulation* algorithm Encap takes as input an encapsulation key and outputs an encapsulation C and a key $k \in \mathcal{G}$. The *blinding* algorithm takes as input an encapsulation key and an encapsulation and outputs a blinded encapsulation $\tilde{C}$ and an unblinding key uk. The *decapsulation* algorithm Decap takes a decapsulation key and a (blinded) encapsulation as input and outputs a (blinded) key $\tilde{k}$. The *unblinding* algorithm takes as input an unblinding key and a blinded key and outputs a key.

The algorithms satisfy the correct decapsulation requirement: When $(ek, dk) \leftarrow \mathsf{KG}_{\mathsf{BKEM}}$, $(C, k) \leftarrow \mathsf{Encap}_{ek}$, $(\tilde{C}, uk) \leftarrow \mathsf{Blind}_{ek}(C)$ and $\tilde{k} \leftarrow \mathsf{Decap}_{dk}(\tilde{C})$, then

$$\mathsf{Unblind}_{uk}(\tilde{k}) = k.$$

Definition 8. Let $\mathsf{BKEM} = (\mathsf{KG}_{\mathsf{BKEM}}, \mathsf{Encap}, \mathsf{Blind}, \mathsf{Decap}, \mathsf{Unblind})$ be a blinded KEM. The *distinguishing advantage* of any adversary $\mathcal{A}$ against BKEM getting r blinded decapsulation samples is

$$\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{BKEM}}(\mathcal{A}, r) = 2\Big|\Pr[\mathbf{Exp}^{\mathsf{ind}}_{\mathsf{BKEM}}(\mathcal{A}, r) = 1] - 1/2\Big|,$$

where the experiment $\mathbf{Exp}^{\mathsf{ind}}_{\mathsf{BKEM}}(\mathcal{A}, r)$ is given in Fig. 4.

```
$\mathbf{Exp}^{\mathsf{ind}}_{\mathsf{BKEM}}(\mathcal{A}, r)$ :
  $b \xleftarrow{\$} \{0,1\}$
  $(ek, dk) \leftarrow \mathsf{KG}_{\mathsf{BKEM}}$
  $(C, k_1) \leftarrow \mathsf{Encap}_{ek}$
  $k_0 \xleftarrow{\$} \mathcal{G}$
  for $j \in \{1, \ldots, r\}$ do
    $(\tilde{C}_j, uk_j) \leftarrow \mathsf{Blind}_{ek}(C)$
    $\tilde{k}_j \leftarrow \mathsf{Decap}_{dk}(\tilde{C}_j)$
  $b' \leftarrow \mathcal{A}(ek, C, k_b, \{(\tilde{C}_j, \tilde{k}_j)\}_{1 \le j \le r})$
  return $b' \overset{?}{=} b$
```

Figure 4: Indistinguishability experiment $\mathbf{Exp}^{\mathsf{ind}}_{\mathsf{BKEM}}(\mathcal{A}, r)$ for a blinded KEM.

Definition 9. Let ek be any public key and let C_0 and C_1 be two encapsulations. Define X_0 and X_1 to be the statistical distribution of the blinded encapsulation output by $\mathsf{Blind}_{ek}(C_0)$ and $\mathsf{Blind}_{ek}(C_1)$, respectively. We say that the blinded KEM is *ϵ-blind* if the statistical distance of X_0 and X_1 is at most ϵ.

Definition 10. Let ek be any public key and let C be an encapsulation of the key k. Let $\tilde{C}$ be a blinded encapsulation of C with corresponding unblinding key uk. We say that the blinded KEM is *rigid* if there is exactly one $\tilde{k}$ such that $\mathsf{Unblind}_{uk}(\tilde{k}) = k$.

We now present two instantiations of blinded KEMs based on well-known hardness assumptions, namely DDH and the RSA problem.

4.1 Construction I: DH-based

We consider the following Diffie-Hellman-based blinded KEM ($\mathsf{DH\text{-}BKEM}$). Let $\mathbb{G}$ be a group of prime order q with generator g and define $\mathsf{DH\text{-}BKEM}$ in Fig. 5.

```
KG_BKEM() :
    $s \xleftarrow{\$} \mathbb{Z}_q^*$
    $ek \leftarrow g^s$
    $dk \leftarrow s$
    return $ek, dk$

Encap_ek :
    $i \xleftarrow{\$} \mathbb{Z}_q^*$
    $C \leftarrow g^i$
    $k \leftarrow ek^i$
    return $C, k$
```

```
Blind_ek(C) :
    $t \xleftarrow{\$} \mathbb{Z}_q^*$
    $\tilde{C} \leftarrow C^t$
    $uk \leftarrow t^{-1} \bmod q$
    return $\tilde{C}, uk$

Decap_dk(\tilde{C}) :
    $\tilde{k} \leftarrow \tilde{C}^{dk}$
    return $\tilde{k}$

Unblind_uk(\tilde{k}) :
    $k \leftarrow \tilde{k}^{uk}$
    return $k$
```

Figure 5: Diffie-Hellman-based blinded KEM (DH-BKEM).

Theorem 1. DH-BKEM *is a* 0*-blind* BKEM *and is rigid. Furthermore, let $\mathcal{A}$ be any adversary against the above construction getting r blinded decapsulation samples. Then there exists an adversary $\mathcal{B}_r$ against* DDH *such that*

$$\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{DH\text{-}BKEM}}(\mathcal{A}, r) \leq \mathbf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{B}_r).$$

The running time of $\mathcal{B}_r$ is essentially the same as the running time of $\mathcal{A}$.

Proof. For any encapsulation, since t is a random number, the blinded encapsulation $\tilde{C}$ output by Blind is uniformly distributed on $\mathbb{G}$. It follows that the construction is 0-blind. In a similar vein, the unblinding procedure is a permutation on the keyspace so the construction is rigid.

Next, consider a tuple (ek, C, k). The reduction $\mathcal{B}_r$ is given in Fig. 6. In the event that (ek, C, k) is a DDH tuple, then $\mathcal{B}_r$ perfectly simulates the input of $\mathcal{A}$ in $\mathbf{Exp}^{\mathsf{ind}}_{\mathsf{DH\text{-}BKEM}}(\mathcal{A}, r)$ when $b = 1$. Otherwise, $\mathcal{B}_r$ perfectly simulates the input of $\mathcal{A}$ in $\mathbf{Exp}^{\mathsf{ind}}_{\mathsf{DH\text{-}BKEM}}(\mathcal{A}, r)$ when $b = 0$. The claim follows.

```
Reduction $\mathcal{B}_r$.
    for $j \in \{1, \ldots, r\}$ do
        $t_j \xleftarrow{\$} \mathbb{Z}_q^*$
        $\tilde{C}_j \leftarrow g^{t_j}$
        $\tilde{k}_j \leftarrow ek^{t_j}$
    $b' \leftarrow \mathcal{A}(ek, C, k, \{(\tilde{C}_j, \tilde{k}_j)\}_{1 \leq j \leq r})$
    return $b'$
```

Figure 6: DDH adversary $\mathcal{B}_r$ playing $\mathbf{Exp}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{B}_r)$, used in the proof of Theorem 1.

□

4.2 Construction II: RSA-based

We consider the following RSA-based blinded KEM (RSA-BKEM). Unlike the above DH-based blinded KEM, this is less suitable for use in key exchange, since generating RSA keys is quite expensive. The scheme needs a hash function $\mathsf{H}_{\mathsf{RSA\text{-}BKEM}}$, and is detailed in Fig. 7.

Just like for the DH-based construction, this scheme is a blinded KEM, it is 0-blind and any adversary against indistinguishability in the random oracle model can be turned into an

```
$\underline{\mathsf{KG}_{\mathsf{BKEM}}():}$
    $p, q, n, e, d \leftarrow \mathsf{RSA.KG}$
    $ek \leftarrow (n, e)$
    $dk \leftarrow (n, d)$
    return $ek, dk$

$\underline{\mathsf{Encap}_{ek}:}$
    $i \xleftarrow{\$} \{1, \ldots, n-1\}$
    $C \leftarrow i^e \bmod n$
    $k \leftarrow \mathsf{H}_{\mathsf{RSA\text{-}BKEM}}(i)$
    return $C, k$

$\underline{\mathsf{Blind}_{ek}(C):}$
    $t \xleftarrow{\$} \{1, \ldots, n-1\}$
    $\tilde{C} \leftarrow (t^e C) \bmod n$
    $uk \leftarrow t^{-1} \bmod n$
    return $\tilde{C}, uk$

$\underline{\mathsf{Decap}_{dk}(\tilde{C}):}$
    $\tilde{k} \leftarrow \tilde{C}^d \bmod n$
    return $\tilde{k}$

$\underline{\mathsf{Unblind}_{uk}(\tilde{k}):}$
    $k' \leftarrow (\tilde{k} uk) \bmod n$
    $k \leftarrow \mathsf{H}_{\mathsf{RSA\text{-}BKEM}}(k')$
    return $k$
```

Figure 7: RSA-based blinded KEM (RSA-BKEM).

adversary against the RSA problem, in a straight-forward way. We omit the proof. Note that this construction is not rigid since any hash collision provides two different values that map to the same k. (Dealing with this would complicate the security proof for little gain.)

5 Offline Assisted Group Key Exchange Protocol

We now describe a generic protocol for cloud-assisted group key exchange using a blinded KEM, and then give a concrete instantiation using our DH-based blinded KEM from Section 4.1. Our scenario consists of the following participants:

- The *initiator* wants to establish a shared key k with a set of responders. First, the initiator I interacts with the server, then the initiator generates a key and "invitation messages" for the responders $\mathsf{R}_1, ..., \mathsf{R}_n$.
- Each *responder* wants to allow the initiator to establish a shared key with him. When responder R_i gets their "invitation message" from the initiator, they will interact with the server to decrypt the shared key.
- The *server* temporarily stores information assisting in the computation of the shared secret key k, until every responder has gotten the key.

A conceptual overview of our construction is given in Fig. 8: the numbering indicates the order in which the phases of the protocol are done. A more diagrammatic overview is provided for the single-responder case in Fig. 9, and the general case is presented in Fig. 10. In these figures and for the rest of this section we will reduce notational overload by writing $\mathsf{Sign}_{\mathsf{R}_j}$ instead of $\mathsf{Sign}_{sk_{\mathsf{R}_j}}$ (and $\mathsf{Enc}_{\mathsf{R}_j}$ instead of $\mathsf{Enc}_{pk_{\mathsf{R}_j}}$ etc.), and allow the reader to infer which type of key is being used from the algorithm in use.

Definition 11. An Offline Assisted Group Key Exchange Protocol (OAGK) is defined in Fig. 10 and is parameterized by the following components. Let

- $\mathsf{BKEM} = (\mathsf{KG}_{\mathsf{BKEM}}, \mathsf{Encap}, \mathsf{Blind}, \mathsf{Decap}, \mathsf{Unblind})$ be a *blinded KEM*,

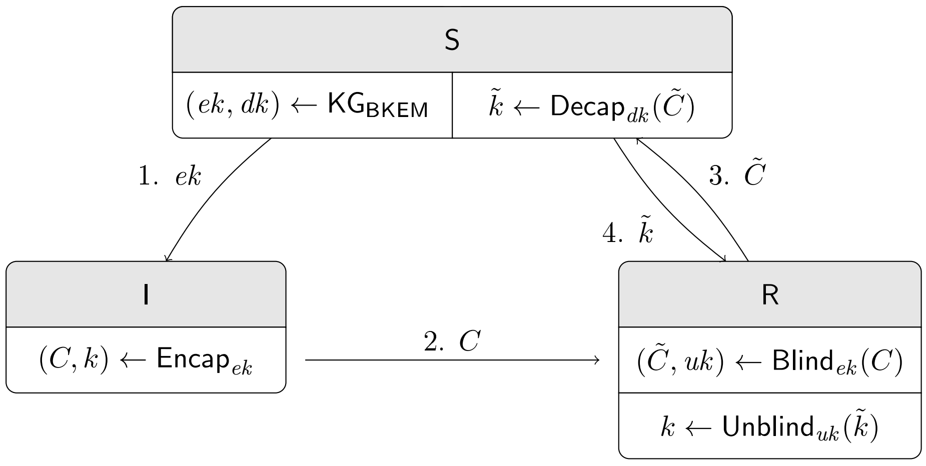
S
$(ek, dk) \leftarrow \mathsf{KG}_{\mathsf{BKEM}}$
$\tilde{k} \leftarrow \mathsf{Decap}_{dk}(\tilde{C})$
1. ek
3. $\tilde{C}$
4. $\tilde{k}$
I
$(C, k) \leftarrow \mathsf{Encap}_{ek}$
2. C
R
$(\tilde{C}, uk) \leftarrow \mathsf{Blind}_{ek}(C)$
$k \leftarrow \mathsf{Unblind}_{uk}(\tilde{k})$

Figure 8: Diagram describing how the group key exchange protocol uses the blinded KEM to do key exchange. For clarity, identities, nonces, session identifiers, key confirmation, public key encryption and digital signatures are omitted.

- $\mathsf{DS} = (\mathsf{KG}_{\mathsf{sig}}, \mathsf{Sign}, \mathsf{Verify})$ be a *signature scheme*,
- $\mathsf{PKE} = (\mathsf{KG}_{\mathsf{pke}}, \mathsf{Enc}, \mathsf{Dec})$ be a *public-key encryption scheme*,
- H be a hash function,
- KDF be a key derivation function.

Note that in our model, we do not have a reveal state query, so there is no need to explicitly erase state information. In a real implementation, making sure that ephemeral and medium-term key material is erased at appropriate times is vital.

5.1 Protocol Security

An adversary against the GKE protocol OAGK plays the game defined in Section 3.2. We need to give a useful bound for its advantage.

Theorem 2. *Consider an adversary $\mathcal{A}$ against the GKE protocol OAGK running with n users, having at most s sessions, each involving at most r responders. Then adversaries $\mathcal{B}_0$, $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{B}_4$ exist, running in essentially the same time as $\mathcal{A}$, such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{ake}}_{\mathsf{OAGK}}(\mathcal{A}) \leq\ & \mathbf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{B}_0) + (n+1)\mathbf{Adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{DS}}(\mathcal{B}_1) \\ & + snr\mathbf{Adv}^{\mathsf{ror\text{-}cca2}}_{\mathsf{PKE}}(\mathcal{B}_2) + s\mathbf{Adv}^{\mathsf{CR}}_{\mathsf{KDF}}(\mathcal{B}_3) + sr\epsilon \\ & + s\mathbf{Adv}^{\mathsf{ind}}_{\mathsf{BKEM}}(\mathcal{B}_4, r) \\ & + \textit{negligible terms.}\end{aligned}$$

We sketch the ideas used in the proof. We need to guess which session the adversary is going to issue the Test query for. If we guess correctly, the game proceeds unchanged. If we guess incorrectly, the game immediately stops, we flip a coin b' and pretend that the adversary output b'. It is clear that the adversary's advantage in this game is now $1/s$ times the original advantage.

We must also handle the situation where the adversary issues a corruption query that would render our chosen session non-fresh. In this case, the game immediately stops, we flip a coin b' and pretend that the adversary output b'. Observe that if we stop for this reason, the adversary could not issue a test query (and our chosen session is now the only session a test query could

I		S		R
		Stage 1:		
Choose nonce $\mathsf{N_I}$				
$\sigma_1 \leftarrow \mathsf{Sign_I}(\mathsf{N_I}, pid)$				
	$\xrightarrow{\mathsf{N_I}, pid, \sigma_1}$			
		Verify σ_1		
		$(ek, dk) \leftarrow \mathsf{KG_{BKEM}}$		
		$\sigma_2 \leftarrow \mathsf{Sign_S}(\mathsf{N_I}, pid, ek)$		
		$sid \leftarrow \mathsf{H}(\mathsf{I}, \mathsf{N_I}, pid, ek)$		
Verify σ_2	$\xleftarrow{(ek, \sigma_2)}$			
$sid \leftarrow \mathsf{H}(\mathsf{I}, \mathsf{N_I}, pid, ek)$				
		Stage 2:		
$(C, k) \leftarrow \mathsf{Encap}_{ek}$				
$k_\mathsf{I} \leftarrow \mathsf{KDF}(''1'', k, sid)$				
$\tau_\mathsf{I} \leftarrow \mathsf{KDF}(''2'', k, sid)$				
$c \leftarrow \mathsf{Enc_R}(C, ek, \tau_\mathsf{I}, sid, pid)$				
$\sigma_3 \leftarrow \mathsf{Sign_I}(c)$				
	$\xrightarrow{c, \sigma_3}$		$\xrightarrow{}$	Verify σ_3
Accept k_I				$(C, ek, \tau_\mathsf{I}, sid, pid) \leftarrow \mathsf{Dec_R}(c)$
		Stage 3:		
				$(\tilde{C}, uk) \leftarrow \mathsf{Blind}_{ek}(C)$
				$\sigma_4 \leftarrow \mathsf{Sign_R}(sid, ek, \tilde{C})$
		Verify σ_4	$\xleftarrow{(sid, \tilde{C}, \sigma_4)}$	
		Verify $\mathsf{R} \in pid$		
		$\tilde{k} \leftarrow \mathsf{Decap}_{dk}(\tilde{C})$		
		$\sigma_5 \leftarrow \mathsf{Sign_S}(sid, \tilde{k})$		
			$\xrightarrow{(sid, \tilde{k}, \sigma_5)}$	Verify σ_5
				$k \leftarrow \mathsf{Unblind}_{uk}(\tilde{k})$
				$k_\mathsf{R} \leftarrow \mathsf{KDF}(''1'', k, sid)$
				$\tau_\mathsf{R} \leftarrow \mathsf{KDF}(''2'', k, sid)$
				$\tau_\mathsf{R} \overset{?}{=} \tau_\mathsf{I}$
				Accept k_R

Figure 9: OAGK protocol showing one responder R.

I running oracle $\prod_{\mathsf{I}}^{\alpha}$ as initiator on input pid:

```
1. Choose random $\mathsf{N_I}$.
2. $\sigma_1 \leftarrow \mathsf{Sign_I}(\mathsf{N_I}, pid)$.
3. Send $(\mathsf{N_I}, pid, \sigma_1)$ to $\mathsf{S}$.
10. Get $(ek, \sigma_2)$ from $\mathsf{S}$.
11. Verify that $\sigma_2$ is $\mathsf{S}$'s signature on
    $(\mathsf{N_I}, pid, ek)$.
12. $sid \leftarrow \mathsf{H}(\mathsf{I}, \mathsf{N_I}, pid, ek)$.
13. $(C, k) \leftarrow \mathsf{Encap}_{ek}$.
14. Session key $k_{\mathsf{I}}^{\alpha} \leftarrow \mathsf{KDF}(''1'', k, sid)$
15. Key confirmation:
    $\tau_{\mathsf{I}}^{\alpha} \leftarrow \mathsf{KDF}(''2'', k, sid)$
16. For every responder $\mathsf{R}_j$ in $pid$, do:
    (a) $c_j \leftarrow \mathsf{Enc}_{\mathsf{R}_j}(C, ek, \tau_{\mathsf{I}}^{\alpha}, sid, pid)$.
    (b) $\sigma_{3,j} \leftarrow \mathsf{Sign_I}(c_j)$.
    (c) Send $(c_j, \sigma_{3,j})$ to $\mathsf{R}_j$.
17. Output $k_{\mathsf{I}}^{\alpha}$.
```

Phase I of S running oracle $\prod_{\mathsf{S}}^{\beta}$ as server on message $(\mathsf{N_I}, pid, \sigma_1)$ from I:

```
4. Verify that $\sigma_1$ is $\mathsf{I}$'s signature on
   $(\mathsf{N_I}, pid)$.
5. $(ek, dk) \leftarrow \mathsf{KG_{BKEM}}$.
6. $\sigma_2 \leftarrow \mathsf{Sign_S}(\mathsf{N_I}, pid, ek)$.
7. $sid \leftarrow \mathsf{H}(\mathsf{I}, \mathsf{N_I}, pid, ek)$.
8. Store $(sid, \mathsf{I}, pid, dk, \emptyset)$.
9. Send $(ek, \sigma_2)$ to $\mathsf{I}$.
```

R_j running oracle $\prod_{\mathsf{R}_j}^{\nu}$ as responder on message $(c_j, \sigma_{3,j})$ from I:

```
18. Verify that $c_j$ is $\mathsf{I}$'s signature on $\sigma_{3,j}$.
19. $(C, ek, \tau_{\mathsf{I}}^{\alpha}, sid, pid) \leftarrow \mathsf{Dec}_{\mathsf{R}_j}(c_j)$.
20. $(\tilde{C}_j, uk_j) \leftarrow \mathsf{Blind}_{ek}(C)$.
21. $\sigma_4 \leftarrow \mathsf{Sign}_{\mathsf{R}_j}(sid, ek, \tilde{C}_j)$.
22. Send $(sid, \tilde{C}_j, \sigma_4)$ to $\mathsf{S}$.
32. Get $(sid, \tilde{k}_j, \sigma_5)$ from $\mathsf{S}$.
33. Verify that $\sigma_5$ is $\mathsf{S}$'s signature on
    $(sid, \tilde{k}_j)$.
34. $k_j \leftarrow \mathsf{Unblind}_{uk_j}(\tilde{k}_j)$.
35. Session key: $k_{\mathsf{R}_j}^{\nu} \leftarrow \mathsf{KDF}(''1'', k_j, sid)$
36. Key confirmation:
    $\tau_{\mathsf{R}_j}^{\nu} \leftarrow \mathsf{KDF}(''2'', k_j, sid)$
    **If** $\tau_{\mathsf{R}_j}^{\nu} = \tau_{\mathsf{I}}^{\alpha}$ **then**
        Accept and output $k_{\mathsf{R}_j}^{\nu}$.
    **else**
        Reject.
```

Phase II of S running oracle $\prod_{\mathsf{S}}^{\beta}$ as server on message $(sid, \tilde{C}_j, \sigma_4)$ from R_j, with stored state $(sid, \mathsf{I}, pid, dk, T)$:

```
23. Lock the state $(sid, \dots)$ until done.
24. Verify that $\sigma_4$ is $\mathsf{R}_j$'s signature on
    $(sid, ek, \tilde{C}_j)$.
25. Verify that $\mathsf{R}_j \in pid$.
26. Verify that $\mathsf{R}_j \notin T$.
27. $\tilde{k}_j \leftarrow \mathsf{Decap}_{dk}(\tilde{C}_j)$.
28. $\sigma_5 \leftarrow \mathsf{Sign_S}(sid, \tilde{k}_j)$.
29. Send $(sid, \tilde{k}_j, \sigma_5)$ to $\mathsf{R}_j$.
30. Let $T' = T \cup \{\mathsf{R}_j\}$.
31. Update the state $(sid, \dots, T)$ to
    $(sid, \dots, T')$.
```

Figure 10: The three roles of the group key exchange protocol. Suppose $\{\mathsf{R}_j\}_{j \in J}$ are the identities of users that I wishes to share a common session key with ($pid = \mathsf{I} \| \{\mathsf{R}_j\}_{j \in J}$). Note that the line numbering indicates the order in which the lines of the various roles are reached during a protocol execution.

be issued for), so the adversary would have no information about b. The probability that the adversary guesses b correctly is therefore unchanged.

Depending on when the server is corrupted (if it is corrupted at all), we need to bound the adversary's advantage in slightly different ways. An upper bound on the adversary's advantage will then be the sum of the two different bounds.

If we suppose that every partnered oracle in our session reached status *accepted* before the server or any player running a partnered oracle is corrupted. In this case, thanks to the signatures and the nonces, the adversary sees at most a blinded KEM public encapsulation key, an encapsulation of a session key, at most r blinded encapsulations of the same session key with corresponding blinded decapsulations. By indistinguishability for the blinded KEM, it follows that the adversary cannot distinguish between the actual encapsulated key and a randomly chosen key, so the adversary has no information about b.

Next, suppose no responder player is ever corrupted. In this case, the adversary (in the worst case) chooses the keys for the blinded KEM, but the public key encryption ensures that the adversary cannot see the actual encapsulation of the key. In other words, the adversary only sees blinded encapsulations of an unknown encapsulation, which reveals little information about the encapsulated key by ϵ-blindness of the blinded KEM. Furthermore, the rigidity of the blinded KEM ensures that every responder can detect an incorrect server response, unless a collision in the key derivation function occurs.

5.2 Proof of Theorem 2

The proof of the theorem consists of a sequence of games.

Game 0

The first game is the game from Def. 6, defining security for our protocol. Let E_0 be the event that the adversary's guess b' equals b from the Test oracle (and let E_i be the corresponding event for Game i). Then

$$\mathbf{Adv}^{\mathsf{ake}}_{\mathsf{OAGK}}(\mathcal{A}) = \Big|\mathbf{Pr}[E_0] - 1/2\Big|. \tag{1}$$

Game 1

We modify the game so that if two server oracles or two initiator oracles ever arrive at the same *sid*, the game stops.

For this to happen, either two oracles must choose the same values for $\mathsf{N_I}$ or ek, or we have found a collision in H. Since there are at most s initiator oracles and server oracles, and s^2 must be small compared to the number of possible nonces or KEM encapsulation keys, the only possible non-negligible term[3] is the possibility of finding a collision in H. We can easily construct a collision-finding algorithm $\mathcal{B}_0$ from $\mathcal{A}$, which shows that

$$\Big|\mathbf{Pr}[E_1] - \mathbf{Pr}[E_0]\Big| \leq \mathbf{Adv}^{\mathsf{CR}}_{\mathsf{H}}(\mathcal{B}_0) + \text{negligible terms}. \tag{2}$$

Game 2

We modify the game so that if any oracle ever verifies a signature from an uncorrupted principal that was not created by another oracle, the game stops.

If this happens, our adversary has produced a forgery for DS. We can trivially produce a forger $\mathcal{B}_1$ for the signature scheme using a standard hybrid argument. Since our n users and the

[3]To be secure, the KEM key generation algorithm must provide sufficient min-entropy to allow us to ignore the possibility that the KEM encapsulation keys collide.

server all have a signing key, we get that

$$\Big|\mathbf{Pr}[E_2] - \mathbf{Pr}[E_1]\Big| \leq (n+1)\mathbf{Adv}^{\mathsf{suf\text{-}cma}}_{\mathsf{DS}}(\mathcal{B}_1). \tag{3}$$

By inspection of the protocol, it is now apparent that in Game 2, the partnering relation on oracles from Def. 4 is an equivalence relation on accepting oracles for which

- every equivalence class whose *pid* contains an uncorrupted initiator contains an initiator oracle; and
- if the server is uncorrupted, every equivalence class contains exactly one server oracle.

Furthermore, this equivalence relation can be extended to an equivalence relation on all oracles, where oracles are related if and only if they have the same *sid*.

Game 3

The next modification we make is to guess which session the adversary will query with the Test query, by choosing a number uniformly at random from $\{1, 2, \ldots, s\}$, identifying the corresponding initiator oracle and guessing that session. If the adversary sends the Test query to this session, we proceed as usual. Otherwise, we stop when the adversary issues the Test query, flip a coin b' and pretend that the adversary output b'.

Since we choose the session randomly, the adversary cannot know anything about which session we choose. It follows that

$$\Big|\mathbf{Pr}[E_2] - 1/2\Big| = s\Big|\mathbf{Pr}[E_3] - 1/2\Big|. \tag{4}$$

Game 4

The next modification we make is that if the adversary every issues a Corrupt query such that our chosen session becomes unfresh, we stop the game, flip a coin b' and pretend that the adversary output b'.

If we never stop the game, this game proceeds exactly as Game 3.

If the adversary corrupts players so that our chosen session becomes unfresh, the adversary cannot ask a Test query of our session. This means that in Game 3, the eventual Test query would go to some other session, which would cause the game to stop and a coin b' to be flipped.

We get that

$$\mathbf{Pr}[E_4] = \mathbf{Pr}[E_3]. \tag{5}$$

Let F be the event that the server is corrupted before our chosen session has completed. Referring to the two clauses in Def. 5, if F is false, the first clause applies, otherwise the second clause applies.

It is easy to show that

$$\Big|\mathbf{Pr}[E_4] - 1/2\Big| \leq \Big|\mathbf{Pr}[E_4|F] - 1/2\Big| + \Big|\mathbf{Pr}[E_4|\neg F] - 1/2\Big|. \tag{6}$$

We can therefore analyse the two cases separately, which we shall proceed to do, using two sequences of games, each beginning with Game 4.

Game 5

We begin by assuming that the server is corrupted, which by the freshness requirements means that the adversary will never get to corrupt the players in our chosen session. We modify the game by having our initiator oracle encrypt random messages instead the real messages. Any responder oracle that receives this exact ciphertext will use values directly from our initiator oracle, instead of decrypting the (nonsense) ciphertext.

We shall now use $\mathcal{A}$ and any difference in $\mathbf{Pr}[E_4|F]$ and $\mathbf{Pr}[E_5]$ to construct an adversary $\mathcal{B}_2$ against multi-user security of public key encryption, as defined in Appendix A.

Our adversary $\mathcal{B}_2$ works as follows:

- It gets encryption keys for the users as input.
- When $\mathcal{B}_2$ must simulate a responder oracle that gets input from a corrupted initiator, it uses its decryption oracle to get the decryption of the ciphertexts.
- When $\mathcal{B}_2$ simulates responders that get input from an uncorrupted initiator, then because we have forbidden signature forgeries, the ciphertext was created by an initiator oracle, so $\mathcal{B}_2$ knows what is inside the ciphertext and does not need to decrypt that ciphertext.
- When the adversary corrupts a principal, $\mathcal{B}_2$ gets the decryption key from its oracle.
- When simulating the initiator oracle of our chosen session, $\mathcal{B}_2$ uses its encryption oracle to encrypt the messages.

We see that if $\mathcal{B}_2$'s encryption oracle encrypts the real messages, $\mathcal{B}_2$ perfectly simulates the situation in Game 4 given F. If $\mathcal{B}_2$'s encryption oracle encrypts random messages, $\mathcal{B}_2$ perfectly simulates the situation in Game 5 given F.

We get that

$$\left|\mathbf{Pr}[E_5|F] - \mathbf{Pr}[E_4|F]\right| \leq nr\mathbf{Adv}^{\mathsf{ror\text{-}cca2}}_{\mathsf{PKE}}(\mathcal{B}_2). \tag{7}$$

Game 6

Next, we modify the responder oracles in our chosen session so that they reject if the unblinded decapsulated key k_j computed in Step 34 does not match the key k computed by the initiator oracle in Step 13.

If a responder oracle rejects in this game, but would not have rejected in the previous game, it has found a collision in KDF. We can therefore construct a collision finder $\mathcal{B}_3$ such that

$$\left|\mathbf{Pr}[E_6|F] - \mathbf{Pr}[E_5|F]\right| \leq \mathbf{Adv}^{\mathsf{CR}}_{\mathsf{KDF}}(\mathcal{B}_3). \tag{8}$$

Game 7

In this game, we modify the responder oracles of our chosen session so that instead of using the encapsulation sent by the initiatior oracle, they create their own encapsulation of a random, independent key using the corrupt server's encapsulation key, blind it and compare the unblinded decapsulation with this key. Instead of computing the key to be output, they simply output the one output by the initiator oracle.

By rigidity, there is exactly one server response that a responder oracle will accept, and this answer depends only on the blinding sent by the responder, not on which encapsulation was used to create the blinding.

It follows by ϵ-blindness that

$$\left|\mathbf{Pr}[E_7|F] - \mathbf{Pr}[E_6|F]\right| \leq r\epsilon. \tag{9}$$

Furthermore, we see that in this game, the adversary has no information about the key chosen by the initiator oracle and later output by the responder oracles. This means that if the adversary asks a Test query for this session the response will be a random key, regardless of the value of b. It follows that

$$\mathbf{Pr}[E_7|F] = 1/2. \tag{10}$$

By equations (7)–(10) we get that

$$\left|\mathbf{Pr}[E_4|F] - 1/2\right| \leq nr\mathbf{Adv}^{\textsf{ror-cca2}}_{\textsf{PKE}}(\mathcal{B}_2) + \mathbf{Adv}^{\textsf{CR}}_{\textsf{KDF}}(\mathcal{B}_3) + r\epsilon. \tag{11}$$

Game 5'

Now we assume that the server is not corrupted until every responder has accepted. We modify the game so that in our chosen session, the initiator oracle ignores the encapsulated key and instead outputs a randomly chosen key. The responder oracles also ignore the key they compute and instead output the key chosen by the initiator oracle.

We can now construct an adversary $\mathcal{B}_4$ against indistinguishability for our blinded KEM. The adversary $\mathcal{B}_4$ gets an encapsulation key, an encapsulation, a key and r pairs of blindings and blinded decapsulations as input. It uses the encapsulation key to simulate the server message to the initiator oracle. It uses the encapsulation to simulate the messages to the responders. And it uses the blindings and blinded decapsulations to simulate the conversations between the responders and the server. Finally, it has the oracles of our chosen session output its input key.

We see that if the key input to $\mathcal{B}_4$ is the real encapsulated key, then $\mathcal{B}_4$ perfectly simulates the situation in Game 4 given $>\neg F$. If the key input to $\mathcal{B}_4$ is a random key, then $\mathcal{B}_4$ perfectly simulates the situation in this game given $\neg F$.

We get that

$$\left|\mathbf{Pr}[E_{5'}|\neg F] - \mathbf{Pr}[E_4|\neg F]\right| = \mathbf{Adv}^{\textsf{ind}}_{\textsf{BKEM}}(\mathcal{B}_4, r). \tag{12}$$

Furthermore, if the adversary asks a $\textsf{Test}$ query for our chosen session in this game, the response will be a random key regardless of the value of b. It follows that

$$\mathbf{Pr}[E_{5'}|\neg F] = 1/2. \tag{13}$$

By equations (12) and (13) we get that

$$\left|\mathbf{Pr}[E_4|\neg F] - 1/2\right| \leq \mathbf{Adv}^{\textsf{ind}}_{\textsf{BKEM}}(\mathcal{B}_4, r). \tag{14}$$

The claim now follows by equations (1)–(6), (11) and (14).

5.3 Instantiating the protocol with the DH blinded KEM

We instantiate the above offline assisted group key exchange protocol $\textsf{OAGK}$ with the DH-based blinded KEM from Section 4.1, the protocol denoted by $\textsf{DH-OAGK}$. In this instantiation, we choose the nonce $\textsf{N}_\textsf{I}$ from the group $\mathbb{G}$.

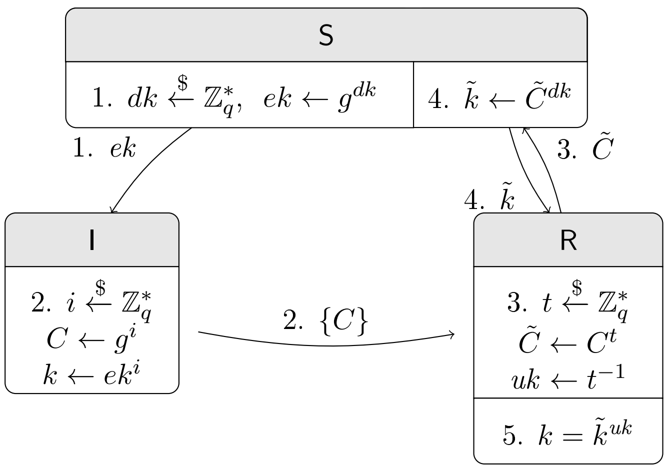
S
1. $dk \xleftarrow{\$} \mathbb{Z}_q^*,\ ek \leftarrow g^{dk}$
4. $\tilde{k} \leftarrow \tilde{C}^{dk}$
1. ek
3. $\tilde{C}$
4. $\tilde{k}$
I
2. $i \xleftarrow{\$} \mathbb{Z}_q^*$ $C \leftarrow g^i$ $k \leftarrow ek^i$
2. $\{C\}$
R
3. $t \xleftarrow{\$} \mathbb{Z}_q^*$ $\tilde{C} \leftarrow C^t$ $uk \leftarrow t^{-1}$
5. $k = \tilde{k}^{uk}$

Figure 11: Running protocol $\textsf{DH-OAGK}$ with one responder, where $\{C\} = \textsf{Enc}_\textsf{R}(C, \cdots)$.

In Fig. 11, we present the core of the resulting protocol (without identities, nonces, session identifiers, key confirmation, authentication and encryption) similar to Fig. 8. We only show one responder.

Thm. 1 and Thm. 2 show that this instantiation is secure.

References

[1] Masayuki Abe, Rosario Gennaro, Kaoru Kurosawa, and Victor Shoup. Tag-KEM/DEM: A new framework for hybrid encryption and a new analysis of Kurosawa–Desmedt KEM. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3494 of *Lecture Notes in Computer Science*, pages 128–146. Springer, 2005.

[2] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *IACR Cryptology ePrint Archive*, 2002:80, 2002.

[3] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014: 34th Annual Cryptology Conference*, pages 480–499. Springer Berlin Heidelberg, 2014.

[4] Emmanuel Bresson and Mark Manulis. Securing group key exchange against strong corruptions. In Masayuki Abe and Virgil D. Gligor, editors, *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008*, pages 249–260. ACM, 2008.

[5] Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 1995.

[6] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *J. Cryptology*, 20(3):265–294, 2007.

[7] Cheng-Kang Chu, Sherman S. M. Chow, Wen-Guey Tzeng, Jianying Zhou, and Robert H. Deng. Key-aggregate cryptosystem for scalable data sharing in cloud storage. *IEEE Trans. Parallel Distrib. Syst.*, 25(2):468–477, 2014.

[8] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *IACR Cryptology ePrint Archive*, 2001:108, 2001.

[9] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 33(1):167–226, 2003.

[10] Alexander W. Dent. A designer's guide to KEMs. In Kenneth G. Paterson, editor, *Cryptography and Coding, 9th IMA International Conference, Cirencester, UK, December 16-18, 2003, Proceedings*, volume 2898 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2003.

[11] Weizheng Gao, Kashi Neupane, and Rainer Steinwandt. Tuning a two-round group key agreement. *Int. J. Inf. Sec.*, 13(5):467–476, 2014.

[12] M. D. Green and I. Miers. Forward secure asynchronous messaging from puncturable encryption. In *2015 IEEE Symposium on Security and Privacy*, pages 305–320, May 2015.

[13] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT key exchange with full forward secrecy. In *EUROCRYPT (3)*, volume 10212 of *Lecture Notes in Computer Science*, pages 519–548, 2017.

[14] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference*, volume 4622 of *Lecture Notes in Computer Science*, pages 553–571. Springer, 2007.

[15] Jonathan Katz and Moti Yung. Scalable protocols for authenticated group key exchange. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 110–125. Springer Berlin Heidelberg, 2003.

[16] Kaoru Kurosawa and Yvo Desmedt. A new paradigm of hybrid encryption scheme. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2004.

[17] István Lám, Szilveszter Szebeni, and Levente Buttyán. Invitation-oriented TGDH: key management for dynamic groups in an asynchronous communication model. In *41st International Conference on Parallel Processing Workshops, ICPPW 2012*, pages 269–276. IEEE Computer Society, 2012.

[18] István Lám, Szilveszter Szebeni, and Levente Buttyán. Tresorium: Cryptographic file system for dynamic groups over untrusted cloud storage. In *41st International Conference on Parallel Processing Workshops, ICPPW 2012*, pages 296–303. IEEE Computer Society, 2012.

[19] Xuefeng Liu, Yuqing Zhang, Boyang Wang, and Jingbo Yan. Mona: Secure multi-owner data sharing for dynamic groups in the cloud. *IEEE Trans. Parallel Distrib. Syst.*, 24(6):1182–1191, 2013.

[20] Moxie Marlinspike and Trevor Perrin. The X3DH key agreement protocol. `https://signal.org/docs/specifications/x3dh/`, November 2016.

A Multi-user Public-Key Encryption Security

In the proof of the main theorems, it is convenient to consider a multi-user variant of public key encryption. The security notion we consider is equivalent to the usual real-or-random security notion for public key encryption. We first explain and define the notion and then prove the relevant theorem.

We consider a *multi-user setting* with n users. All users use PKE, each user U_i keeps their own secret decryption key sk_i and all public encryption keys are assumed to be known to the public (and thus all algorithms).

For our security analysis we define the *adversary's capacity.* The adversary is given all public keys and can ask for challenge encryptions of any (valid) message under different public keys. In a chosen-ciphertext attack the adversary is allowed to ask for decryptions of arbitrary ciphertexts, except for those that would allow a trivial win.

We also give the adversary the ability to corrupt a user, that is, obtain the secret key of the corrupted user. In order to prevent trivial wins, we must restrict this capability to users for which the adversary has not yet asked for challenge encryptions. (This is a fundamental restriction for ordinary public-key encryption. For other notions such as puncturable encryption or non-committing encryption, this restriction could be somewhat relaxed.)

We now define *real-or-random indistinguishability* for a multi-user public-key encryption scheme under chosen-ciphertext attack and corruption attack ($\mathsf{mu\text{-}ror\text{-}cca2}$): an adversary cannot distinguish encryptions of chosen plaintexts, possibly encrypted under different public keys, from the encryptions of equal-length random strings, encrypted under the same public keys.

In the definition of the security experiment we employ a list $\mathsf{F}_{\mathsf{LIST}}$ of of forbidden ciphertext, a list $\mathsf{U}_{\mathsf{LIST}}$, and a corrupted user $\mathsf{C}_{\mathsf{LIST}}$ to prevent trivial wins.

Remark 1. *We now describe the restrictions on our adversaries that we enforce by using* $\mathsf{F}_{\mathsf{LIST}}$*,* $\mathsf{U}_{\mathsf{LIST}}$ *and* $\mathsf{C}_{\mathsf{LIST}}$*. If the adversary asks its real-or-random (*$\mathcal{O}.\mathsf{RoR}$*) challenge oracle for some corrupted user (that belongs to* $\mathsf{C}_{\mathsf{LIST}}$*), the oracle will return encryptions of real messages. If the adversary asks for decryptions of some ciphertext that it received from its* $\mathcal{O}.\mathsf{RoR}$ *oracle, the adversary will obtain nothing (to stop trivial wins). In a* $\mathsf{Corrupt}$ *query, the adversary cannot reveal the secret key of some user in* $\mathsf{U}_{\mathsf{LIST}}$*, since the challenge oracle has returned encryptions under their key (which means that revealing the key would allow the adversary to win trivially by decrypting the challenge ciphertext).*

Definition 12. Let $\mathsf{PKE} = (\mathsf{KG}_{\mathsf{pke}}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. Then the $\mathsf{mu\text{-}ror\text{-}cca2}$ advantage of an adversary $\mathcal{A}$ against PKE is defined as

$$\mathbf{Adv}^{(t,\,n,\,c)\text{-}\mathsf{mu\text{-}ror\text{-}cca2}}_{\mathsf{PKE}}(\mathcal{A}) = 2\left|\mathbf{Pr}[\mathbf{Exp}^{(t,\,n,\,c)\text{-}\mathsf{mu\text{-}ror\text{-}cca2}}_{\mathsf{PKE}}(\mathcal{A}) = 1] - \frac{1}{2}\right|.$$

where n is the number of users, c the maximal number of corrupted users and t the maximal number of challenge ciphertexts the adversary can receive, respectively. The experiment $\mathbf{Exp}^{(t,\,n,\,c)\text{-}\mathsf{mu\text{-}ror\text{-}cca2}}_{\mathsf{PKE}}(\mathcal{A})$ is given in Fig. 12.

The following result describes the relationship between the $\mathsf{mu\text{-}ror\text{-}cca2}$ notion and the usual $\mathsf{ror\text{-}cca2}$ notion.

Theorem 3. *Let* $\mathsf{PKE} = (\mathsf{KG}_{\mathsf{pke}}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme. Let* $\mathcal{A}$ *be an adversary against* PKE *under adaptive chosen ciphertext attack and corruption attack in the multi user setting, running with* n *users. Suppose* c *is the maximal number of corrupted users,* t *is the maximal number of challenge ciphertexts the adversary can receive. Then there exists an adversary* $\mathcal{B}$ *against* PKE *under adaptive chosen ciphertext attack in the single user setting, such that*

$$\mathbf{Adv}^{(t,\,n,\,c)\text{-}\mathsf{mu\text{-}ror\text{-}cca2}}_{\mathsf{PKE}}(\mathcal{A}) \leq nt\mathbf{Adv}^{\mathsf{ror\text{-}cca2}}_{\mathsf{PKE}}(\mathcal{B}).$$

```
$\mathbf{Exp}^{(t,n,c)\text{-mu-ror-cca2}}_{\mathsf{PKE}}(\mathcal{A})$ :
    $b \xleftarrow{\$} \{0,1\}$
    $\mathsf{F_{LIST}}, \mathsf{U_{LIST}}, \mathsf{C_{LIST}} \leftarrow \emptyset$
    for $j \in \{1,\dots,r\}$ do
        $(sk_j, pk_j) \leftarrow \mathsf{KG_{pke}}$
        $\overrightarrow{pk} \leftarrow \overrightarrow{pk} \cup pk_j$
    $b' \leftarrow \mathcal{A}^{\mathcal{O}.\mathsf{RoR}_b, \mathcal{O}.\mathsf{Dec}, \mathcal{O}.\mathsf{Corrupt}}(\overrightarrow{pk})$
    return $b' \overset{?}{=} b$

$\mathcal{O}.\mathsf{Corrupt}(pk)$
    if $pk \in \mathsf{U_{LIST}}$ then
        return $\perp$
    $\mathsf{C_{LIST}} \leftarrow \mathsf{C_{LIST}} \cup \{pk\}$
    return $sk$
```

```
$\mathcal{O}.\mathsf{RoR}_b(pk, m)$ :
    if $pk \in \mathsf{C_{LIST}}$ then
        return $c \leftarrow \mathsf{Enc}_{pk}(m)$
    $m_1 \leftarrow m$
    $m_0 \xleftarrow{\$} \mathcal{M}_{pk}$
    $c \leftarrow \mathsf{Enc}_{pk}(m_b)$
    $\mathsf{U_{LIST}} \leftarrow \mathsf{U_{LIST}} \cup \{pk\}$
    $\mathsf{F_{LIST}} \leftarrow \mathsf{F_{LIST}} \cup \{c\}$
    return $c$

$\mathcal{O}.\mathsf{Dec}(pk, c)$
    if $c \in \mathsf{F_{LIST}}$ then
        return $\perp$
    $m \leftarrow \mathsf{Dec}_{sk}(c)$
    return $m$
```

Figure 12: The experiment defining (t, n, c)-mu-ror-cca2 security for a public-key encryption scheme $\mathsf{PKE} = (\mathsf{KG_{pke}}, \mathsf{Enc}, \mathsf{Dec})$.

A.1 Proof of Theorem 3

The proof is in three parts. The first part is a straight-forward hybrid argument, reducing the number of key pairs to one. The second part shows that when we only consider a single key pair, we can disregard the corruption oracle. And finally, the third part is again a straight-forward hybrid argument reducing the number of challenge encryptions to one. This completes the argument, since $(1, 1, 0)$-mu-ror-cca2 is the same as ror-cca2.

Part 1. We first prove that there exists an adversary $\mathcal{A}_1$ against PKE under adaptive chosen ciphertext attack and corruption attack in the single user setting, such that

$$\mathbf{Adv}^{(t,n,c)\text{-mu-ror-cca2}}_{\mathsf{PKE}}(\mathcal{A}) \leq n\mathbf{Adv}^{(t,1,1)\text{-mu-ror-cca2}}_{\mathsf{PKE}}(\mathcal{A}_1). \tag{15}$$

Proof. We use a hybrid argument with $n + 1$ hybrid games, counting from 0. For corrupted users, $\mathcal{O}$.RoR will always encrypt real messages. In the ith hybrid game the challenge oracle $\mathcal{O}$.RoR will encrypt real messages for the ith first public keys. For the remaining $n - i$ public keys, the challenge oracle will encrypt random messages.

An adversary's advantage is bounded by n times the average distinguishing advantage for the same adversary against two consecutive hybrid games.

Now we use a (t, n, c)-adversary $\mathcal{A}$ to create a $(t, 1, 1)$-adversary $\mathcal{A}_1$ against the scheme, and prove that this new adversary has the same advantage as the average distinguishing advantage for $\mathcal{A}$ against two consecutive hybrid games. The adversary $\mathcal{A}_1$ is given in Fig. 13.

If $\mathcal{A}_1$'s challenge oracle always encrypts the real message, then $\mathcal{A}_1$ perfectly simulates the ith hybrid game for $\mathcal{A}$. Likewise, if $\mathcal{A}_1$'s challenge oracle always encrypts random messages, then $\mathcal{A}_1$ perfectly simulates the $i - 1$th hybrid game for $\mathcal{A}$.

When $\mathcal{A}_1$ has chosen i, and thereby the two hybrid games to potentially simulate, its advantage is exactly equal to the distinguishing advantage of $\mathcal{A}$ for the two consecutive hybrid games chosen. Since $\mathcal{A}_1$ chooses i uniformly at random, the advantage of $\mathcal{A}_1$ is exactly equal to the average distinguishing advantage of $\mathcal{A}$ against two consecutive hybrid games.

The claim follows.

□

```
Reduction $\mathcal{A}_1$.
  $i \xleftarrow{\$} \{1, 2, \ldots, n\}$
  $\mathsf{F_{LIST}}, \mathsf{U_{LIST}}, \mathsf{C_{LIST}} \leftarrow \emptyset$
  receive $pk_i$
  for $j \in \{1, \ldots, n\} \setminus \{i\}$ do
    $(sk_j, pk_j) \leftarrow \mathsf{KG_{pke}}$
  $\overrightarrow{pk} \leftarrow (pk_1, pk_2, \ldots, pk_n)$
  $b' \leftarrow \mathcal{A}^{\mathcal{O}.\mathsf{RoR}_b, \mathcal{O}.\mathsf{Dec}, \mathcal{O}.\mathsf{Corrupt}}(\overrightarrow{pk})$
  return $b'$

$\mathcal{O}.\mathsf{Corrupt}(pk_j)$
  if $pk_j \in \mathsf{U_{LIST}}$ then
    return $\perp$
  if $pk_j = pk_i$ then
    $\underline{sk \leftarrow \mathcal{O}.\mathsf{Corrupt}(pk_i)}$
  else
    $sk \leftarrow sk_j$
  $\mathsf{C_{LIST}} \leftarrow \mathsf{C_{LIST}} \cup \{pk_j\}$
  return $sk$
```

```
$\mathcal{O}.\mathsf{RoR}_b(pk_j, m)$ :
  if $pk_j \in \mathsf{C_{LIST}}$ then
    return $c \leftarrow \mathsf{Enc}_{pk_j}(m)$
  $m^\$ \xleftarrow{\$} \mathcal{M}_{pk}$
  if $pk_j = pk_i$ then
    $\underline{c \leftarrow \mathcal{O}.\mathsf{RoR}(pk_i, m)}$
  if $j < i$ then
    $c \leftarrow \mathsf{Enc}_{pk_j}(m)$
  if $j > i$ then
    $c \leftarrow \mathsf{Enc}_{pk_j}(m^\$)$
  $\mathsf{U_{LIST}} \leftarrow \mathsf{U_{LIST}} \cup \{pk_j\}$
  $\mathsf{F_{LIST}} \leftarrow \mathsf{F_{LIST}} \cup \{c\}$
  return $c$

$\mathcal{O}.\mathsf{Dec}(pk_j, c)$
  if $c \in \mathsf{F_{LIST}}$ then
    return $\perp$
  if $pk_j = pk_i$ then
    $\underline{m \leftarrow \mathcal{O}.\mathsf{Dec}(pk_i, c)}$
  if $j \neq i$ then
    $m \leftarrow \mathsf{Dec}_{sk_j}(c)$
  return $m$
```

Figure 13: Reduction $\mathcal{A}_1$ playing $\mathbf{Exp}_{\mathsf{PKE}}^{(t,1,1)\text{-}\mathsf{mu\text{-}ror\text{-}cca2}}(\mathcal{A}_1)$, used in proof of (15).

Part 2. We now prove that there exists an $(t, 1, 0)$-mu-ror-cca2 adversary $\mathcal{A}_2$ against PKE such that

$$\mathbf{Adv}_{\mathsf{PKE}}^{(t,1,1)\text{-}\mathsf{mu\text{-}ror\text{-}cca2}}(\mathcal{A}_1) = \mathbf{Adv}_{\mathsf{PKE}}^{(t,1,0)\text{-}\mathsf{mu\text{-}ror\text{-}cca2}}(\mathcal{A}_2). \tag{16}$$

Proof. We first note that if $\mathcal{A}_1$ calls its corruption oracle on its single public key, it has no way to get any information about b, so its advantage is 0.

The adversary $\mathcal{A}_2$ runs $\mathcal{A}_1$. It forwards any $\mathcal{O}$.RoR and $\mathcal{O}$.Dec queries from $\mathcal{A}_1$ to its own oracles. If $\mathcal{A}_1$ queries its corruption oracle, $\mathcal{A}_2$ stops, flips a fair coin b' and outputs b'.

If $\mathcal{A}_1$ does not query its corruption oracle, $\mathcal{A}_2$ proceeds exactly as $\mathcal{A}_1$ and wins with exactly the same probability. Furthermore, if $\mathcal{A}_1$ does query its corruption oracle, $\mathcal{A}_2$ does not proceed exactly as $\mathcal{A}_1$, but it wins with exactly the same probability.

Let E be the event that $\mathcal{A}_1$ wins, E' the event that $\mathcal{A}_2$ wins, and let F be the event that $\mathcal{A}_1$ queries its corruption oracle, while F' is the probability that $\mathcal{A}_2$ flips a fair coin to determine its result. Note that $\mathbf{Pr}[F] = \mathbf{Pr}[F']$ by definition, and $\mathbf{Pr}[E|F] = \mathbf{Pr}[E'|F']$ and $\mathbf{Pr}[E|\neg F] = \mathbf{Pr}[E'|\neg F]$ by the above paragraphs. Then we have

$$\begin{aligned}\mathbf{Pr}[E] &= \mathbf{Pr}[E|F]\mathbf{Pr}[F] + \mathbf{Pr}[E|\neg F]\mathbf{Pr}[\neg F]\\ &= \mathbf{Pr}[E'|F']\mathbf{Pr}[F'] + \mathbf{Pr}[E'|\neg F']\mathbf{Pr}[\neg F']\\ &= \mathbf{Pr}[E'].\end{aligned}$$

The claim follows. □

Part 3. We now prove, again using a standard hybrid argument, that there exists an $(1, 1, 0)$-mu-ror-cca2 adversary $\mathcal{A}_3$ such that

$$\mathbf{Adv}_{\mathsf{PKE}}^{(t,1,0)\text{-}\mathsf{mu\text{-}ror\text{-}cca2}}(\mathcal{A}_2) \leq t\mathbf{Adv}_{\mathsf{PKE}}^{(1,1,0)\text{-}\mathsf{mu\text{-}ror\text{-}cca2}}(\mathcal{A}_3). \tag{17}$$

Proof. Again, we have a hybrid argument with $t+1$ hybrid games, counting from 0. In the ith hybrid game, the challenge oracle $\mathcal{O}.\mathsf{RoR}$ will encrypt the real message for the first i queries, and then encrypt random messages for the remaining $t-i$ queries.

An adversary's advantage is bounded by t times the average distinguishing advantage for the same adversary against two consecutive hybrid games.

Now we use a $(t,1,0)$-mu-ror-cca2 adversary $\mathcal{A}_2$ to create a $(1,1,0)$-mu-ror-cca2 adversary $\mathcal{A}_3$ against the scheme, and prove that this new adversary has the same advantage as the average distinguishing advantage for $\mathcal{A}_2$ against two consecutive hybrid games. The adversary $\mathcal{A}_3$ is given in Fig. 14

If $\mathcal{A}_3$'s challenge oracle encrypts the real message, then $\mathcal{A}_3$ perfectly simulates the ith hybrid game for $\mathcal{A}_2$. Likewise, if $\mathcal{A}_3$'s challenge oracle encrypts a random message, then $\mathcal{A}_3$ perfectly simulates the $i-1$th hybrid game for $\mathcal{A}_2$.

When $\mathcal{A}_3$ has chosen i, and thereby two hybrid games to potentially simulate, its advantage is exactly equal to the distinguishing advantage of $\mathcal{A}_2$ for the two consecutive hybrid games chosen. Since $\mathcal{A}_3$ chooses i uniformly at random, the advantage of $\mathcal{A}_3$ is exactly equal to the average distinguishing advantage of $\mathcal{A}_2$ against two consecutive hybrid games.

The claim follows. □

```
Reduction $\mathcal{A}_3$.
  receive $pk$
  $\mathsf{F_{LIST}} \leftarrow \emptyset$
  $b' \leftarrow \mathcal{A}_2^{\mathcal{O}.\mathsf{RoR}_b, \mathcal{O}.\mathsf{Dec}, \mathcal{O}.\mathsf{Corrupt}}(pk)$
  return $b'$

$\mathcal{O}.\mathsf{Dec}(pk, c)$
  if $c \in \mathsf{F_{LIST}}$ then
    return $\perp$
  $m \leftarrow \mathcal{O}.\mathsf{Dec}(c)$
  return $m$
```

```
$\mathcal{O}.\mathsf{RoR}_b(pk, m_j)$ :
  $m^\$ \xleftarrow{\$} \mathcal{M}_{pk}$
  if $j = i$ then
    $c \leftarrow \mathcal{O}.\mathsf{RoR}(m_i)$
  if $j < i$ then
    $c \leftarrow \mathsf{Enc}_{pk}(m_j)$
  if $j > i$ then
    $c \leftarrow \mathsf{Enc}_{pk}(m^\$)$
  $\mathsf{F_{LIST}} \leftarrow \mathsf{F_{LIST}} \cup \{c\}$
  return $c$
```

Figure 14: The reduction $\mathcal{A}_3$ from $(t,1,0)$-mu-ror-cca2 to $(1,1,0)$-mu-ror-cca2 used to prove (17).

Now we observe that a $(1,1,0)$-mu-ror-cca2 adversary against the scheme is simply an ror-cca2 adversary, and the theorem follows from equations (15)–(17).