Security-Preserving Asymmetric Protocol Encapsulation

Raphael C.-W. Phan and Serge Vaudenay

Ecole Polytechnique Fédérale de Lausanne (EPFL),
CH-1015 Lausanne, Switzerland
{raphael.phan,serge.vaudenay}@epfl.ch

Abstract. Query-response based protocols between a client and a server such as SSL, TLS, SSH are asymmetric in the sense that the querying client and the responding server play different roles, and for which there is a need for two-way linkability between queries and responses within the protocol. We are motivated by the observation that though results exist in other related contexts, no provably secure scheme has been applied to the setting of client-server protocols, which differ from conventional communications on the above points. We show how to secure the communication of queries and responses in these client-server protocols in a provably secure setting. In doing so, we propose a new primitive: a query-response encapsulation scheme; we give an instantiation, and we demonstrate how this primitive can be used for our purpose. In our proof of secure encapsulation, we show how to preserve the notion of "local-security".

1 Introduction

In this paper, we show how to secure the communication of generic client-server type protocols. These are protocols which involve query and response messages for arbitrary number of rounds, and are asymmetric in the sense that parties on either side of the protocol have differing roles.

Asymmetric protocols. Client-server type protocols exist in many settings [8, 9, 17–19, 22, 25], e.g. in networks where clients are connected to application servers, database servers, file servers, or mail servers etc. A typical example of such a protocol is a client browser running on a personal computer interacting with a web server sitting on a remote machine. In this setting, it is desired that at the worst case, having the client access the server is at least as secure as just having the client run in isolation. This in fact shows that access to the server does not compromise the security of the client. We denote this notion as *security preservation.*

Security preservation. By *local security*, we mean the security of the client when it is in isolation. In the generic sense, this can be formally modelled as a game following some rules, played between an adversary and a judge challenger who determines if the adversary succeeded in winning the game.

We want to show that when the client is additionally allowed access to a server, then this can be done such that the local security of the client is preserved, i.e. additional server access does not have adverse affects on the client's security.

In the ideal case, interaction with the server is not observable by the adversary. In practice, we show how this can be done via a form of asymmetric protocol encapsulation that preserves local security. This result applies to any asymmetric protocol of arbitrary rounds.

Examples. One example of an asymmetric client-server protocol is database systems where the client system $\mathcal{S}$ represents the application querying the database server $\mathcal{O}$. So far, existing security models for databases [12, 14] are different from our context because they only consider security on the database server $\mathcal{O}$ side. Our focus is on the local security on the client system $\mathcal{S}$ side and how this is affected by the $\mathcal{S} \leftrightarrow \mathcal{O}$ interaction. For instance, let us assume a company $\mathcal{A}$ making queries to a low-cost airline company's online database. A competitor $\mathcal{B}$ to $\mathcal{A}$ who suspects that $\mathcal{A}$ is interested in some type of event can forge a flyer for a fake event at some given location and date, check on the air ticket cost, send the fake event advertisement to $\mathcal{A}$. When $\mathcal{B}$ sees that some request to the airline company was made by $\mathcal{A}$, it can look at the prices again and see if the price increased because $\mathcal{A}$ has just taken the low cost seat. If the link between $\mathcal{A}$ and the airline company is perfectly secure, $\mathcal{B}$ does not even see when the request is made. But when the link is implemented in practice e.g. using SSL through an insecure channel such as the Internet, $\mathcal{B}$ gets this private information. Here, we can see that the encapsulated system is insecure even though the implementations of the client and the server are secure.

Another example is an EMV protocol [11] where the credit-card payment terminal $\mathcal{S}$ makes queries to a bank $\mathcal{O}$ to verify a message authentication code (MAC) issued by a credit card. Here, the response of the bank is a YES/NO reply. Clearly, this system can reach pretty good security if the client-server communication is made through a perfectly secure link. When instantiated in practice with an insecure channel, things could still easily go wrong even though some encapsulation of the channel is applied that achieves some strong cryptographic notion of confidentiality and authentication. For instance, an adversary could try to replay a positive response from the bank without breaking the underlying cryptographic primitives. Indeed, if the primitives do not provide strong linkability between the query and its response, a fake credit card can easily be used for payment. This example shows that some extra cryptographic property must be assumed on the encapsulation scheme used to secure the client-server interaction.

In this paper, our goal is to make explicit this property, to show how to achieve it, and to show that local security can be preserved while moving from a secure (ideal) channel to an insecure but encapsulated one.

Related work. The seminal introduction of public-key encryption by Diffie and Hellman [10] initiated serious work in the public domain into solving the problem of securely encapsulating insecure channels. In fact, the Diffie-Hellman key exchange of [10] was the first to allow the establishment of secure channels without pre-sharing any common secrets between parties, thus solving the key distribution problem inherent in conventional symmetric encryption. With a key exchange protocol, parties can securely establish common shared keys that are then used to encrypt subsequent communications. In a sense, major research has focused on initializing the secure channel hence the study of key exchange protocols, in contrast to the subsequent step after initialization: the encapsulation of the communication channel.

In fact, the study of how to encapsulate an insecure channel is an interesting problem in itself because it is often that a channel needs not only be encapsulated in terms of secrecy, authentication and integrity, but also resistant against other forms of malicious manipulation e.g. reordering or replaying of messages, exploitation of one legitimate party to answer challenges from the other party etc. Indeed, carefully analyzing attacks applied to key exchange protocols reveals some of these problems too, because essentially, it can be seen by looking at the internals of a key exchange protocol that it is a sequence of message transmissions over an insecure channel.

Work initiated by Canetti and Krawczyk [5, 6, 18, 20, 21] showed how secure encapsulated channels can be obtained as a by-product of secure key establishment, for the particular setting where each transmitted message is independent of other messages in the encapsulated channel.

Their line of work differs on several points from the context of client-server communication considered in this paper. Client-server communication is asymmetric i.e. each party on either side has a different role, so it matters which side the party is on. More importantly, this leads to adversely different security requirements e.g. there is a need for two-way linkability between the transmitted query and the received response, a requirement not needed for conventional secure *independent-message* channels in [5, 6, 18, 20, 21]. Furthermore, we need only *semi-authenticated* communication, i.e. we consider both ends separately and where it is only required to authenticate the server side (see [17]). Or alternatively see [19, 22] for a kind of asymmetric authentication where variable levels of authentication apply depending on which side the party is on.

In essence, the motivation behind our considering semi-authentication is that our local security consideration essentially reduces the security of the client-server interaction to the security on the client side. This also models the client-server systems in practice where a single server provides equal access of its resources to any client.

A different line of work related to client-server systems is the study of provably secure schemes *within* the server e.g. provable security of the database server in the case of database systems [12, 14].

Our contributions. Our main result is a generic notion for provably secure encapsulation of asymmetric client-server protocols. To the best of our knowledge, our paper is the first work to treat this problem in a provably secure setting. Our results generalize to asymmetric protocols with arbitrary number of rounds. To this end, we give a generic construction for a new primitive, so called the *query-response encapsulation mechanism (Q/REM)*, that provides confidentiality and semi-authentication of query and response messages. We also give an example instantiation and prove its security.

This Q/REM primitive allows to design client-server protocol communications in a modular fashion. On the one hand, specific details of a client-server protocol are abstracted away during the security proof of encapsulation. On the other, once it is established that a secure encapsulated channel is in place between client and server, the intrinsic details of the protocol can be designed in this ideal encapsulated setting without any further need to treat requirements for confidentiality, integrity and authentication.

In some sense, this modular abstraction bears resemblance to the approach in [5], where key establishment protocols are constructed assuming the existence of encapsulated (only in the sense of authentication) channels, while at a different design level it is studied how protocols designed in the encapsulated model relate to the practical model by the use of *authenticating encapsulators*.

In our proof of secure asymmetric protocol encapsulation, we show that the local security on the client side is preserved provided that the communication protocol is locally secure when the adversary knows when the communication happens and its length, and that the server is protected against replay attacks or is a stateless oracle.

2 A Generic Construction for Query-Response Encapsulation

We give here a generic construction for a query/response encapsulation mechanism (Q/REM) primitive and define its corresponding security notions of indistinguishability and authentication. This result in itself is of independent interest; for instance we know of no related primitive that achieves the notion of authentication. The QEM is a regular message transmission primitive yielding a symmetric key which can be re-used to encapsulate a response. The REM re-uses this key. It is then shown how this Q/REM primitive can be used to build an asymmetric encapsulation protocol.

2.1 Query/Response Encapsulation Mechanism (Q/REM)

To be precise, the Q/REM primitive is given in Definition 1.

Definition 1 (Q/REM).

A *query/response encapsulation mechanism (Q/REM)* is a 5-tuple of algorithms.

$\langle pk, sk\rangle \leftarrow QEM.Key(1^\lambda)$:	a probabilistic polynomial time (PPT) algorithm taking as input a security parameter λ, and returns a pair $\langle pk, sk\rangle$ of matching public and private keys.
$\langle K, A\rangle \leftarrow QEM.Enc(pk, q)$:	a PPT algorithm taking as input a public key pk and query q; outputs an ephemeral key/encapsulation pair $\langle K, A\rangle$.
$\langle K, q\rangle \leftarrow QEM.Dec(sk, A)$:	a deterministic polynomial time (DPT) algorithm taking as input a private key sk and an encapsulation A; outputs an ephemeral key K and a decapsulated query q, or a special symbol $\perp$ implying A was invalid.
$B \leftarrow REM.Enc(K, i, r)$:	a PPT algorithm taking as input an ephemeral key K, counter i and a response r, and outputs an encapsulation B.
$r \leftarrow REM.Dec(K, i, B)$:	a DPT algorithm taking as input an ephemeral key K, counter i and an encapsulation B, and outputs a decapsulated response r or a special symbol $\perp$ implying B was invalid.

For *completeness*, it is required that for any $\langle pk, sk\rangle$ output by $QEM.Key(\cdot)$ and for any $\langle K, A\rangle$ output by $QEM.Enc(pk, q)$, then $QEM.Dec(sk, A) = \langle K, q\rangle$; and furthermore, for any output B by $REM.Enc(K, i, r)$, then $REM.Dec(K, i, B) = r$.

Security notions. The security of Q/REM is captured by two notions: *indistinguishability* in a IND-CCA vein (Definition 2), and *authentication* (Definition 3).

Definition 2 (Indistinguishability of Q/REM). *Let IND-CCA-QREM denote the security notion of indistinguishability for Q/REM against CCA adversaries [23]. This notion is described by considering the following attack scenario, modeled as a game between an adversary and a challenger with access to Q/REM algorithms as oracles:*

1. *$QEM.Key(1^\lambda)$ is run to generate the public pk and private key sk for the protocol, and pk is given to the adversary.*
2. *The adversary may perform some $OQEM.Dec(A)$ queries, where $OQEM.Dec(\cdot)$ is a QEM decapsulation oracle, that is, it returns the output by $QEM.Dec(sk, A)$.*
3. *The adversary selects a query q^* and submits it to the challenger. The challenger chooses $b \in \{0, 1\}$, lets $q_0 = q^*$, and q_1 set to a random query of same length. He runs $QEM.Enc(pk, q_b)$ to obtain $\langle K, A^* \rangle$. The output A^* is returned to the adversary.*
4. *The adversary does as in Step 2 except that the query $OQEM.Dec(A^*)$ is not allowed. It can further make $OIREM.Enc(i, r)$ queries, where $OIREM.Enc(\cdot)$ is an oracle which either outputs $B \leftarrow REM.Enc(K, i, r)$ if $b = 0$, or outputs $B \leftarrow REM.Enc(K, i, r^*)$ for some random r^* of same length otherwise. (Note that b is the same as in the previous step.)*
5. *The adversary outputs a guess $\tilde{b} \in \{0, 1\}$.*

Let $\pi_{\mathtt{Q/REM}}$ be a Q/REM protocol and let $\mathcal{A}$ be the adversary. The advantage of $\pi_{\mathtt{Q/REM}}$ for adversary $\mathcal{A}$, is defined as:

$$Adv^{\mathtt{IND-CCA}}_{\mathcal{A},\pi_{\mathtt{Q/REM}}}(\lambda) = |\Pr[\tilde{b} = b] - 1/2|.$$

$\pi_{\mathtt{Q/REM}}$ is $(\varepsilon(\lambda), c(\lambda))$-secure in the sense of IND-CCA-QREM if $Adv^{\mathtt{IND-CCA}}_{\mathcal{A},\pi_{\mathtt{Q/REM}}}(\lambda)$ is less than $\varepsilon(\lambda)$ for any adversary $\mathcal{A}$ of complexity bounded by $c(\lambda)$. It is simply IND-CCA-QREM-secure if for any polynomially bounded $c(\lambda)$ there exists a negligible $\varepsilon(\lambda)$ such that it is $(\varepsilon(\lambda), c(\lambda))$-secure in the sense of IND-CCA-QREM.

Definition 3 (Authentication of Q/REM). *Let AUTH-CCA-QREM denote the security notion of authentication for Q/REM against CCA adversaries. This notion is as follows:*

1. *$QEM.Key(1^\lambda)$ is run to generate the public pk and private key sk for the protocol, and pk is given to the adversary.*
2. *The adversary may perform some $OQEM.Dec(\cdot)$ queries.*
3. *The challenger gets one query q chosen by the adversary, runs $QEM.Enc(pk, q)$ to obtain $\langle K, A^* \rangle$, and returns A^* to the adversary.*
4. *The adversary does as in Step 2 except that the call $OQEM.Dec(A^*)$ is not allowed. She can further do many $OREM.Enc(\cdot)$ queries. Namely, querying $OREM.Enc(i, r)$ makes the challenger run $REM.Enc(K, i, r)$ to obtain B^* which is returned to the adversary.*
5. *The adversary outputs $\langle i, B \rangle$. She succeeds if $REM.Dec(K, i, B)$ is valid but B is not equal to any output from a $OREM.Enc(\cdot)$ query from the previous step.*

Let $\pi_{\mathtt{Q/REM}}$ be a Q/REM protocol and let $\mathcal{A}$ be the adversary. The advantage of $\pi_{\mathtt{Q/REM}}$ for adversary $\mathcal{A}$, is defined as:

$$Adv^{\mathtt{AUTH-CCA}}_{\mathcal{A},\pi_{\mathtt{Q/REM}}}(\lambda) = \Pr[\mathcal{A} \text{ succeeds}].$$

$\pi_{\mathtt{Q/REM}}$ is $(\varepsilon(\lambda), c(\lambda))$-secure in the sense of AUTH-CCA-QREM if $Adv^{\mathtt{AUTH-CCA}}_{\mathcal{A},\pi_{\mathtt{Q/REM}}}(\lambda)$ is less than $\varepsilon(\lambda)$ for any adversary $\mathcal{A}$ of complexity bounded by $c(\lambda)$. It is simply AUTH-CCA-QREM-secure if for any polynomially bounded $c(\lambda)$ there exists a negligible $\varepsilon(\lambda)$ such that it is $(\varepsilon(\lambda), c(\lambda))$-secure in the sense of AUTH-CCA-QREM.

Definition 4. *A $(\varepsilon_{sem}, \varepsilon_{auth}, c)$-secure Q/REM is a Q/REM which is (ε_{sem}, c)-secure in the sense of IND-CCA-QREM and (ε_{auth}, c)-secure in the sense of AUTH-CCA-QREM. A secure Q/REM is an IND-CCA-QREM-secure and AUTH-CCA-QREM-secure Q/REM.*

2.2 Asymmetric Protocol Encapsulation with Q/REM

We can secure any 2-party asymmetric protocol between a Client and a Server for arbitrary number of rounds, by using a Q/REM. For this protocol, given input q the Client system should obtain the output $r = \mathcal{O}(q)$ from the Server oracle $\mathcal{O}$. We construct an asymmetric (query-response) encapsulation protocol π_{QR}. A trusted communication channel (e.g. using a trusted third party) is required at the initialization stage to authenticate the public key pk of the Server to the Client.

Initialization: Server runs $QEM.Key(1^\lambda)$, obtains $\langle pk, sk\rangle$, and stores $\langle pk, sk\rangle$. Client obtains pk in a trusted way.

Query Generation: Upon input query q, Client runs $QEM.Enc(pk, q)$, obtaining $\langle K, A\rangle$, where K is an ephemeral (secret) key and A is the encapsulated query. Client sends A to the Server.

Response Generation: Server runs $QEM.Dec(sk, A)$ and obtains $\langle K, q\rangle$, where K is the ephemeral key, or obtains $\perp$ if A is invalid and therefore aborts. Server inputs the query q to its internal oracle $\mathcal{O}$ and obtains the response $r = \mathcal{O}(q)$. Server runs $REM.Enc(K, 1, r)$ and obtains B the encryption of response r under secret key K. Server sends B to the Client.

Response Verification: Client runs $REM.Dec(K, 1, B)$ and obtains decapsulated response r or $\perp$ if B is invalid and therefore aborts. Otherwise output r.

Values of $i > 1$ in REM will be used in protocols with several rounds.

2.3 Instantiating Q/REMs

The generic Q/REM primitive in Definition 1 can in fact be instantiated in several ways using public-key encryption or signcryption with one-pass authenticated encryption, or symmetric encryption with message authentication etc.

For completeness, we give a concrete instantiation example based on a secure KEM [7] and secure Authenticated Encryption (AE) [2, 24], and prove its security. Here, we use AE instead of conventional DEM [7] because we need the authentication property and the ability to include a header to re-use a key. Basically, our QEM instantiation is a KEM together with an AE with header 0; subsequent REMs use AE with header set to the counter i.

To the best of our knowledge, this is the first time that a KEM+AE composition is applied for the *authenticity* context whereas previous work used KEM+DEM [7, 1] or KEM+AE [16] for achieving indistinguishability, and that for symmetric independent-message communication.

Q/REM Instantiation based on KEM and AE.

1. $\langle pk, sk\rangle \leftarrow QEM.Key(1^\lambda)$: This is exactly $KEM.Key(1^\lambda)$.
2. $\langle K, A\rangle \leftarrow QEM.Enc(pk, q)$: First, $\langle K, C_0\rangle \leftarrow KEM.Enc(pk)$. This generates the encapsulation in C_0 of an AE key K. Then, $C \leftarrow AE.Enc(K, 0, q)$. The output of $QEM.Enc(\cdot)$ is A where $A = C_0||C$ is the encapsulation output of QEM.
3. $\langle K, q\rangle \leftarrow QEM.Dec(sk, A)$: Parse $A = C_0||C$. Then, $K \leftarrow KEM.Dec(sk, C_0)$: Given as input a private key sk and a key encapsulation C_0, it outputs the decapsulated secret key K, or $\perp$ if the encapsulation C_0 is invalid. Finally, $q \leftarrow AE.Dec(K, 0, C)$: Given as input a secret key K and a message ciphertext C, it outputs the decrypted query q, or outputs $\perp$ if the ciphertext C is not authenticated. The output of $QEM.Dec(\cdot)$ is q or $\perp$.
4. $B \leftarrow REM.Enc(K, i, r)$: Do $B \leftarrow AE.Enc(K, i, r)$: Given a secret key K, a counter i and a response r, it outputs the authenticated ciphertext B. The output of $REM.Enc(\cdot)$ is B.
5. $r \leftarrow REM.Dec(K, i, B)$: Do $r \leftarrow AE.Dec(K, i, B)$ or $\perp$ if B is not authenticated. The output of $REM.Dec(\cdot)$ is r or $\perp$.

The following result shows that this Q/REM instantiation is secure; a detailed proof is in Appendix A.

Theorem 1. *If KEM is IND-CCA-secure, and AE is AE-secure, then the Q/REM construction described above is secure.*

3 Provably Secure Encapsulation

We prove here how to secure asymmetric protocols with arbitrary rounds, involving two parties: a client system $\mathcal{S}$ issuing an initial query q (resp. subsequent query denoted r_t) to a server oracle $\mathcal{O}$ who replies with response r (resp. r_{t+1}). Both parties interact with an (a priori untrusted) environment $\mathcal{E}$. We term as "system" the $\mathcal{S} \leftrightarrow \mathcal{O}$ combination which lives with environment $\mathcal{E}$.

Ideally, the environment does not see the $\mathcal{S} \leftrightarrow \mathcal{O}$ interaction, nor even see when it occurs. In reality, this interaction is necessarily insecure, for which we will show here how to securely model this ideal interaction by encapsulation. An adversary against $\mathcal{S}$ is a malicious environment who interacts with the system, for which we can tell if he succeeds in following the rules of a local game Γ. The notion of "rule of a local game" Γ should be understood as a given judge algorithm Γ who determines from the interactions between $\mathcal{E}$ and $\mathcal{S}$ only (and not other interactions e.g. between $\mathcal{E}$ and $\mathcal{O}$) if the attack succeeded, i.e if the adversary won the game against $\mathcal{S}$. (See Fig. 1.) Not every security notions for a system can be expressed locally. When it is possible to express security this way, it is quite a strong security property because we are saying that having the extra interaction with $\mathcal{O}$ does not give the adversary any additional advantage over just having interaction with $\mathcal{S}$ alone. Our goal is to show how to preserve this local security when the ideal $\mathcal{S} \leftrightarrow \mathcal{O}$ interaction is in reality instantiated with a secure protocol encapsulation.

Definition 5 (Local security). *An oracle-system $\mathcal{S} \leftrightarrow \mathcal{O}$ in a hostile environment $\mathcal{E}$ is locally secure w.r.t. a game Γ i.e. (ε, c)-Γ-secure, if no adversary $\mathcal{E}$ with complexity less than c succeeds in winning game Γ with probability larger than ε.*

We assume the complexity of the adversary includes that of the environment $\mathcal{E}$ and system $\mathcal{S}$.

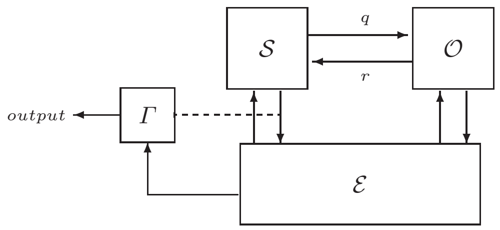

Fig. 1. An oracle-system $\mathcal{S}$ interacting with an oracle $\mathcal{O}$ and an environment $\mathcal{E}$

Local security applies in the examples given in Section 1. For database queries, Γ checks if $\mathcal{E}$ guessed a bit $f(e)$ telling whether Company $\mathcal{A}$ is interested in some chosen event e. For EMV, Γ checks if the payment system accepted a fake credit card.

Here, our notion of local security ideally assumes that the $\mathcal{S} \leftrightarrow \mathcal{O}$ interaction is done through a perfectly secure channel. In reality, this channel is an insecure one going through $\mathcal{E}$, for which we would like to secure by means of encapsulation while preserving local security. See Fig. 2 for the resultant encapsulate-system.

Definition 6 (Encapsulate-system). *Given an oracle-system $\mathcal{S} \leftrightarrow \mathcal{O}$ and a Q/REM encapsulation scheme, we define an* encapsulate-system $\mathcal{S}^{enc} \leftrightarrow \mathcal{O}$ *as follows:*

- *Initially, $\mathcal{S}^{enc}$ runs QEM.Key, stores sk and outputs pk to the environment.*
- *When the environment inputs something to $\mathcal{S}^{enc}$, this is forwarded to a simulated $\mathcal{S}$, and when $\mathcal{S}$ outputs something for the environment, this is output by $\mathcal{S}^{enc}$ to the environment.*
- *When $\mathcal{S}$ outputs a query q for the oracle $\mathcal{O}$, this executes $QEM.Enc(pk, q) \to \langle K, A\rangle$ and a message "QUERY A" is returned to the environment by a special $\mathcal{I}_{\mathcal{S}}$ port. The $\mathcal{I}_{\mathcal{S}}$ port stores $\langle K, 0\rangle$.*
- *When a message "QUERY A'" is input by the environment to $\mathcal{S}^{enc}$ by a special $\mathcal{I}_{\mathcal{O}}$ port, this executes $QEM.Dec(sk, A') \to \langle K', q'\rangle$, queries $\mathcal{O}$ with q' and gets r' in return, executes $REM.Enc(K', 1, r') \to B'$ and returns the message "RESPOND B'" to the environment by the same port. The $\mathcal{I}_{\mathcal{O}}$ port stores $\langle K', 1\rangle$. If QEM.Dec does not work, $\mathcal{S}^{enc}$ aborts.*
- *When "RESPOND B" is input by environment to $\mathcal{S}^{enc}$ by a special $\mathcal{I}_{\mathcal{S}}$ port storing $\langle K, t\rangle$, this increments t and runs $REM.Dec(K, t, B) \to r$ and returns r to $\mathcal{S}$. If REM.Dec does not work, $\mathcal{S}^{enc}$ aborts.*
- *Subsequently, when $\mathcal{S}$ outputs a response r_t for the oracle and the $\mathcal{I}_{\mathcal{S}}$ port stores $\langle K, t\rangle$, this post-increments t, runs $REM.Enc(K, t, r) \to B$ and a message "RESPOND B_t" is returned to the environment by the same port.*
- *When "RESPOND B_t" is input by environment to $\mathcal{S}^{enc}$ by a special $\mathcal{I}_{\mathcal{O}}$ port storing $\langle K', t\rangle$, this post-increments t and runs $REM.Dec(K', t, B) \to r'$, sends r' to $\mathcal{O}$ to get r''; increments t again; runs $REM.Enc(K', t, r'') \to B'$ and returns the message "RESPOND B'_{t+1}" to the environment by the same port.*

Any notion of Γ-security on $\mathcal{S} \leftrightarrow \mathcal{O}$ naturally extends to $\mathcal{S}^{enc} \leftrightarrow \mathcal{O}$ by making Γ ignore all communication through the $\mathcal{I}_{\mathcal{S}}$ and $\mathcal{I}_{\mathcal{O}}$ ports.

Definition 7 (Encapsulate-security). *An oracle-system $\mathcal{S} \leftrightarrow \mathcal{O}$ is (ε, c)-Γ*-encapsulate-secure *relative to a Q/REM encapsulation scheme if its encapsulate-system $\mathcal{S}^{enc} \leftrightarrow \mathcal{O}$ is (ε, c)-Γ-secure.*

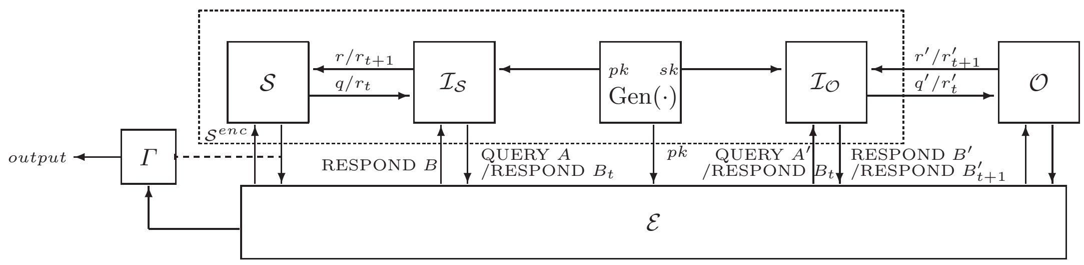

Fig. 2. An encapsulate-system including oracle-system $\mathcal{S}$, interfaces $\mathcal{I}$ and generator Gen

The notion of encapsulate-security captures the fact that local security (with respect to a local game Γ) is preserved when the oracle-system in the ideal case as per Fig. 1 is instantiated in reality with an encapsulate-system as per Fig. 2.

In reality, encapsulation necessary leaks side information:

- [Timing] the point in time at which $\mathcal{S}$ makes its queries and gets its responses, and
- [Length] the length of those query/response messages.

In fact, strong notions of encryption secrecy e.g. IND-CCA also assume that length information is necessarily leaked.

Thus, to prove that encapsulation preserves local security in the practical setting, we must assume that this kind of leakage is harmless. To this end, we define the notion of "toll-security", which captures the oracle-system in the setting similar to Fig. 1 but for which the above side information are inevitably leaked. This is the system that our encapsulate-system needs to emulate.

Definition 8 (Toll-system). *Given an oracle-system $\mathcal{S} \leftrightarrow \mathcal{O}$, we define a* toll-system $\mathcal{S}^{toll} \leftrightarrow \mathcal{O}$ *(see Fig. 3) as follows.*

- *When the environment inputs something to $\mathcal{S}^{toll}$, this is forwarded to a simulated $\mathcal{S}$, and when $\mathcal{S}$ outputs something for the environment, this is output by $\mathcal{S}^{toll}$ to the environment.*
- *When $\mathcal{S}$ outputs a query q (resp. a subsequent response r_t) to the oracle, a message "$\mathcal{S} : |q|$" (resp. "$\mathcal{S} : |r_t|$") with the length $|q|$ of q (resp. $|r_t|$ of r_t) is output by $\mathcal{S}^{toll}$ to the environment by a special $\mathcal{I}$ port and the delivery of q (resp. r_t) to $\mathcal{O}$ through the secure channel is stalled.*
- *When the delivery of q (resp. r_t) is released, $\mathcal{O}$ is queried with q (resp. r_t) to get the response r (resp. r_{t+1}). An "$\mathcal{O} : |r|$" (resp. "$\mathcal{O} : |r_{t+1}|$") message with the length $|r|$ of r (resp. $|r_{t+1}|$ of r_{t+1}) is output by $\mathcal{S}^{toll}$ for the environment by a special $\mathcal{I}$ port and the delivery of r (resp. r_{t+1}) is stalled.*
- *When the environment inputs a "GO $\mathcal{S}$" (resp. "GO $\mathcal{O}$") signal to $\mathcal{S}^{toll}$ by a special $\mathcal{I}$ port, the delivery of the message to $\mathcal{O}$ (resp. $\mathcal{S}$) is released.*

Definition 9 (Toll-security). *An oracle-system $\mathcal{S} \leftrightarrow \mathcal{O}$ is (ε, c)-Γ-*toll-secure *if its toll-system $\mathcal{S}^{toll} \leftrightarrow \mathcal{O}$ is (ε, c)-Γ-secure.*

Note that many secure oracle-systems are trivially toll-insecure, i.e. they become insecure when side information are leaked. For instance, given a secure signature scheme in the random oracle model, we define a new signature scheme for which the signature algorithm on message m first computes some Boolean $\mathsf{bit}(m, sk)$ function depending on the private key sk and, if the bit is 0, does a query to the random oracle with an empty input, otherwise does nothing, then simulates the original signature scheme. Obviously, this is equivalent to the original scheme in the random oracle model, but leaks $\mathsf{bit}(m, sk)$ to the environment in the toll-system variant. This helps the adversary to reconstruct sk after a few queries when bit is well chosen.

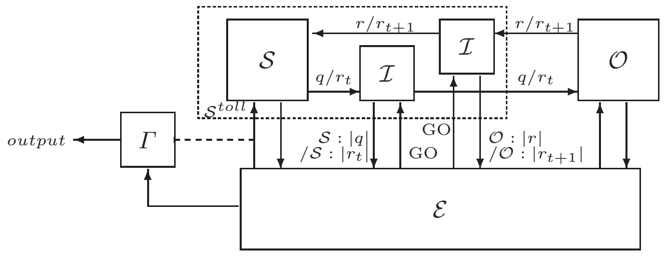

Fig. 3. A toll-system including the oracle-system $\mathcal{S}$ and the interface $\mathcal{I}$

To prove that encapsulation preserves local security with *stateful* oracles, extra treatment is needed: the oracle must be immune to replay attacks.

Definition 10 (Immunity to replay attacks). *An oracle $\mathcal{O}$ is immune to replay attacks, if duplicate queries do not modify the state of $\mathcal{O}$ and produce responses with either an error message, or the same distribution and same length as the previous query.*

This requirement is necessary otherwise a toll-secure system may not be encapsulate-secure if the oracle is stateful and allows repeated queries. To motivate this assumption, we consider that we have a stateful oracle who simulates a perfect random oracle except when a query is repeated, in which case the oracle answers with a constant value (e.g. 0) to the repeated query. An adversary can actually repeat any query from the system and make the system accept answers to the second query, which provokes responses (in this case a constant value) to be perfectly predictable. In a case where queries by the Enquirer are unknown to $\mathcal{E}$, we can have loss of local security when encapsulating the protocol. This can be avoided with an oracle $\mathcal{O}$ that is immune to replay attacks as per Definition 10. For instance, a stateless oracle or an oracle who repeats the same response to repeating queries without modifying its state would satisfy this condition.

Finally, $\mathcal{S}$ must not make concurrent queries to $\mathcal{O}$. Concretely, $\mathcal{S}$ never sends a new message to $\mathcal{O}$ if he is still waiting for a response. Otherwise, $\mathcal{E}$ could get advantage in modifying the order in which they are sent to a stateful oracle.

We now state our main result about secure asymmetric protocol encapsulation. The proof of Theorem 2 is in Appendix B, and shows that if for any Q/REM secure in the sense defined in section 2, then the encapsulate-system is computationally indistinguishable from the ideal toll-system.

Theorem 2. *For any oracle-system $\mathcal{S} \leftrightarrow \mathcal{O}$ in which $\mathcal{O}$ is immune to replay attacks and $\mathcal{S}$ initiates at most Q (non-concurrent) protocol sessions with $\mathcal{O}$, for any notion of local security Γ, there exists some μ such that if $\mathcal{S} \leftrightarrow \mathcal{O}$ is (ε, c)-Γ-toll-secure and if the Q/REM scheme is $(\varepsilon_{sem}, \varepsilon_{auth}, c)$-secure, then $\mathcal{S} \leftrightarrow \mathcal{O}$ is $(Q(\varepsilon_{sem} + \varepsilon_{auth}) + \varepsilon, c - \mu)$-$\Gamma$-encapsulate-secure.*

4 Conclusion

We have put forth a generic notion of security for asymmetric (query-response) protocols, that is set up to permit leakages of timing and message length to closely model types of information leakages in practice. In doing so, we proposed a generic construction of a provably secure query-response encapsulation scheme that can be used to this respect, and our results apply to any arbitrary-round asymmetric protocol.

In our proof of encapsulation, we have also modeled the time when a query is made because otherwise it may not properly capture the leakage of side information that would break the system. We prove the security of encapsulation with the assumption

1. that our underlying primitives are secure,
2. that the query-response protocol remains secure when the existence and time of the query and response, and their length leak,
3. that the server is a stateless machine or a stateful one that is immune to replay attacks.

We further demonstrate that the last two conditions are necessary.

References

1. M. Abe, R. Gennaro, K. Kurosawa and V. Shoup, "Tag-KEM/DEM: A New Framework for Hybrid Encryption and a New Analysis of Kurosawa-Desmedt KEM," *Advances in Cryptology – EUROCRYPT '05*, LNCS 3494, pp. 128–146, 2005.
2. M. Bellare and C. Namprempre. "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," *Advances in Cryptology – ASIACRYPT '00*, LNCS 1976, pp. 531–545, 2000.
3. M. Bellare and P. Rogaway, "Random Oracles are Practical: a Paradigm for Designing Efficient Protocols," *Proc. ACM-CCS '93*, pp. 62–73, 1993.
4. R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," *Proc. IEEE FOCS '01*, pp. 136–145, 2001. Full version available at IACR ePrint Archive, `http://eprint.iacr.org/2000/067`.
5. R. Canetti and H. Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels," *Advances in Cryptology – EUROCRYPT '01*, LNCS 2045, pp. 453–474, 2001.
6. R. Canetti and H. Krawczyk, "Universally Composable Notions of Key Exchange and Secure Channels," *Advances in Cryptology – EUROCRYPT '02*, LNCS 2332, pp. 337–351, 2002.
7. R. Cramer and V. Shoup, "Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack," *SIAM Journal of Computing*, Vol. 33, No. 1, pp. 167–226, 2004.
8. T. Dierks and C. Allen, "The TLS Protocol: Version 1.0," RFC 2246, January 1999.
9. T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol: Version 1.1," RFC 4346, April 2006.
10. W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, pp. 644–654, 1976.
11. EMVCo, LLC, "EMV 4.1 Specifications," June 2004. Available online at `http://www.emvco.com/specifications.asp?show=3`.
12. S. Evdokimov, M. Fischmann and O. Gunther, "Provable Security for Outsourcing Database Operations," *Proc. IEEE ICDE '06*, pp. 117, 2006.
13. A.O. Freier, P. Karlton and P.C. Kocher, "The SSL Protocol: Version 3.0," Internet Draft, March 1996.
14. T. Ge and S. Zdonik, "Fast, Secure Encryption for Indexing in a Column-Oriented DBMS," *Proc. IEEE ICDE '07*, pp. 676-685, 2007.
15. S. Goldwasser, S. Micali and R.L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," *SIAM Journal of Computing*, Vol. 17, No. 2, pp. 281–308, 1988.
16. D. Hofheinz and E. Kiltz, "Secure Hybrid Encryption from Weakened Key Encapsulation," *Advances in Cryptology – CRYPTO '07*, LNCS 4622, pp. 553–571, 2007.
17. C. Krauß, F. Stumpf and C. Eckert, "Detecting Node Compromise in Hybrid Wireless Sensor Networks using Attestation Techniques," *Proc. ESAS '07*, LNCS 4572, pp. 203–217, 2007.

18. H. Krawczyk, "The Order of Encryption and Authentication for Protecting Communications (or: How Secure is SSL?)," *Advances in Cryptology – CRYPTO '01*, LNCS 2139, pp. 310–331, 2001.
19. A. Leung and C.J. Mitchell, "Ninja: Non Identity Based, Privacy Preserving Authentication for Ubiquitous Environments," *Proc. UbiComp '07*, LNCS 4717, pp. 73–90, 2007.
20. W. Nagao, Y. Manabe and T. Okamoto, "A Universally Composable Secure Channel Based on the KEM-DEM Framework," *Proc. TCC '05*, LNCS 3378, pp. 426–444, 2005.
21. W. Nagao, Y. Manabe and T. Okamoto, "A Universally Composable Secure Channel Based on the KEM-DEM Framework," *IEICE Trans. Fund. Electronics, Communications & Computer Sciences*, Vol. E89-A, No. 1, pp. 28–38, 2006.
22. A. Pashalidis and C.J. Mitchell, "Single Sign-On using Trusted Platforms," *Proc. ISC '03*, LNCS 2851, pp. 54–68, 2003.
23. C. Rackoff and D. Simon, "Non-interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack," *Advances in Cryptology – CRYPTO '91*, LNCS 576, pp. 433–444, 1991.
24. P. Rogaway and T. Shrimpton, "A Provable-Security Treatment of the Key-Wrap Problem," *Advances in Cryptology – EUROCRYPT '06*, LNCS 4004, pp. 373–390, 2006.
25. T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol," RFC 4253, January 2006.

Acknowledgement

We thank the anonymous referees for comments that contributed to the clarity of this paper. The second author thanks Tom Shrimpton for discussions on [24] and on AE in general.

A Proof of Theorem 1

To obtain the proof for Theorem 1, we need to recall the definitions of KEM, AE and their corresponding security notions.

KEM. A key encapsulation mechanism (KEM) [7] is given by the triple of algorithms $KEM.Key(1^\lambda), KEM.Enc(pk)$ and $KEM.Dec(sk, C_0)$, where:

1. $\langle pk, sk\rangle \leftarrow KEM.Key(1^\lambda)$: this is a PPT algorithm that takes a security parameter 1^λ and returns a pair $\langle pk, sk\rangle$ of matching public and private keys.
2. $\langle K, C_0\rangle \leftarrow KEM.Enc(pk)$: this is a PPT algorithm that takes as input a public key pk and outputs a key/ciphertext pair $\langle K, C_0\rangle$.
3. $K \leftarrow KEM.Dec(sk, C_0)$: this is a deterministic polynomial time algorithm that takes as input the private key sk and ciphertext C_0, and outputs key K or a special symbol $\perp$ implying the ciphertext was invalid.

It is required that *completeness* be fulfilled, i.e. for all (pk, sk) output by $KEM.Key(\cdot)$, and for all C_0 output by $KEM.Enc(pk)$, then $KEM.Dec(sk, C_0) = K$.

The security notion of IND-CCA-KEM is as follows:

1. $KEM.Key(\cdot)$ is run to generate the public pk and private key sk for the protocol, and pk is given to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ generates some ciphertext queries and sends them to the challenger. He calls the decryption oracle $KEM.Dec(\cdot)$ who decrypts them and the results are returned to $\mathcal{A}$.
3. The challenger runs $KEM.Enc(\cdot)$ to generate $\langle K^*, C_0^*\rangle$. He generates a random string $\tilde{K}$ where $|\tilde{K}| = |K^*|$. He chooses $b \in \{0,1\}$. If $b = 0$, he outputs $\langle K^*, C_0^*\rangle$, otherwise he outputs $\langle \tilde{K}, C_0^*\rangle$.
4. $\mathcal{A}$ generates further ciphertext queries, but cannot query the challenge ciphertext C_0^*.

5. $\mathcal{A}$ outputs a guess $\tilde{b} \in \{0,1\}$.

Let $\mathit{\Pi}_{\mathtt{KEM}}$ be a KEM. The advantage of $\mathit{\Pi}_{\mathtt{KEM}}$ for adversary $\mathcal{A}$, is defined as:

$$Adv^{\mathtt{IND-CCA}}_{\mathcal{A},\mathit{\Pi}_{\mathtt{KEM}}}(\lambda) = |\Pr[\tilde{b} = b] - 1/2|.$$

$\mathit{\Pi}_{\mathtt{KEM}}$ is secure if $Adv^{\mathtt{IND-CCA}}_{\mathcal{A},\mathit{\Pi}_{\mathtt{KEM}}}(\lambda)$ is negligible for any PPT adversary $\mathcal{A}$.

AE. A non-deterministic Authenticated-Encryption (AE) scheme is composed of a key generator and symmetric encryption which use an extra input called the "header" h. [24] follows a series of papers on authenticated-encryption, notably [2] and proposes an all-in-one definition which is adapted below. The decryption algorithm may return $\perp$ if the ciphertext is not consistent with the header. An AE is ε-secure if for any distinguisher with access to two oracles: the left and the right oracle, the advantage for distinguishing the two sets of oracles below is at most ε.

– The left oracle is an encryption oracle. The right oracle is a decryption oracle. Both oracles are first set up with a random key prior to any query.
– The left oracle is a random generator. The right oracle returns $\perp$ except for (y, h) queries such that y was previously output by the left oracle with a (x, h) query: in this case, the right oracle returns x.

Proof of Theorem 1.

Proof. Indistinguishability. We first reduce the IND-CCA-QREM game, i.e. we construct an adversary $\mathcal{A}_{\mathtt{KEM}}$ against IND-CCA-KEM using an adversary $\mathcal{A}$ that breaks IND-CCA-QREM. When $\mathcal{A}_{\mathtt{KEM}}$ receives pk, this is forwarded to $\mathcal{A}$. Queries $A = C_0||C$ for $OQEM.Dec(A)$ from $\mathcal{A}$ to $\mathcal{A}_{\mathtt{KEM}}$ are answered by forwarding C_0 to $KEM.Dec(\cdot)$ oracle. The returned K is used by $\mathcal{A}_{\mathtt{KEM}}$ to perform $AE.Dec(K, 0, C)$ and result q is returned to $\mathcal{A}$ as $\langle K, q\rangle$.

During the challenge stage, $\mathcal{A}$ chooses q^* and sends to $\mathcal{A}_{\mathtt{KEM}}$. Meanwhile, $KEM.Enc(pk)$ is run by the KEM challenger in challenge phase of the IND-CCA-KEM game to generate the key-encapsulation pair $\langle K^\dagger, C_0^*\rangle$. Then depending on a flipped bit b, either the computed $K^\dagger$ or a random one $\tilde{K}$ is returned as K^* with C_0^* to $\mathcal{A}_{\mathtt{KEM}}$ as the challenge $K^*||C_0^*$. $\mathcal{A}_{\mathtt{KEM}}$ sets $q_0 = q^*$ and selects a random q_1 of same length. Then $\mathcal{A}_{\mathtt{KEM}}$ flips a bit b' and performs $C^* = AE.Enc(K^*, 0, q_{b'})$. $A^* = C_0^*||C^*$ is returned to $\mathcal{A}$ as a challenge.

After this, queries for $OQEM.Dec(A)$ by $\mathcal{A}$ are answered similar to before, except for $A = C_0||C$ with $C_0 = C_0^*$ (and $C \neq C^*$). In such a case, $\mathcal{A}_{\mathtt{KEM}}$ runs $AE.Dec(K^*, 0, C)$ directly since it knows the decapsulated K^*. Queries for $OIREM.Enc(i, r)$ can be easily answered by $\mathcal{A}_{\mathtt{KEM}}$ since it knows K^* and b', i.e. it sets $r_0 = r$ and selects a random r_1. Then it returns $B = AE.Enc(K^*, i, r_{b'})$ to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs a bit $\tilde{b}$, $\mathcal{A}_{\mathtt{KEM}}$ outputs $\tilde{b} \oplus b'$.

Let IND-CCA-QREM* be the modified game which behaves just like the original IND-CCA-QEM game except that instead of the secret K^* used in $AE.Enc(\cdot)$ and $AE.Dec(\cdot)$, a completely independent and random key $\tilde{K}$ is used. Note that for $b = 0$, $\mathcal{A}$ plays the IND-CCA-QREM game with $\mathcal{A}_{\mathtt{KEM}}$. When $b = 1$, $\mathcal{A}$ is essentially playing the IND-CCQ-QREM* game with $\mathcal{A}_{\mathtt{KEM}}$. Let X and X^* be events that $\mathcal{A}$ wins the IND-CCA-QREM and IND-CCA-QREM* games respectively. We obtain

$$\begin{aligned}\mathtt{Pr}[\mathcal{A}_{\mathtt{KEM}}\ \mathtt{wins}] - \frac{1}{2} &= \mathtt{Pr}[\tilde{b} \oplus b' = b] - \frac{1}{2} = \frac{1}{2}(\mathtt{Pr}[\tilde{b} = b'|b = 1] - \mathtt{Pr}[\tilde{b} = b'|b = 0]) \\ &= \frac{1}{2}(\mathtt{Pr}[X^*] - \mathtt{Pr}[X]).\end{aligned}$$

Since $|\mathtt{Pr}[\mathcal{A}_{\mathtt{KEM}}\ \mathtt{wins}] - \frac{1}{2}| \leq Adv_{\pi_{\mathtt{KEM}}}^{\mathtt{IND-CCA}}(\lambda)$, thus $|\mathtt{Pr}[X^*] - \mathtt{Pr}[X]| \leq 2Adv_{\pi_{\mathtt{KEM}}}^{\mathtt{IND-CCA}}(\lambda)$.

We now reduce the IND-CCA-QREM* game to the AE game, i.e we use an adversary $\mathcal{A}$ against IND-CCA-QREM* to construct an adversary $\mathcal{A}_{\mathtt{AE}}$ against AE security. Note that KEM is no longer relevant in the IND-CCA-QREM* game because the key $\tilde{K}$ used to key AE is random instead of being generated by KEM.

$\mathcal{A}_{\mathtt{AE}}$ runs $KEM.Key(\cdot)$ obtaining $\langle pk, sk\rangle$. pk is given to $\mathcal{A}$. Queries $A = C_0||C$ for $OQEM.Dec(A)$ from $\mathcal{A}$ to $\mathcal{A}_{\mathtt{AE}}$ are trivially answered by $\mathcal{A}_{\mathtt{AE}}$ since it has sk to run $QEM.Dec(sk, C_0)$ obtaining K which is then used in $AE.Dec(K, 0, C)$. The result q is returned to $\mathcal{A}$ as $\langle K, q\rangle$ for $q \neq \perp$.

During the challenge stage, $\mathcal{A}$ chooses q^* and sends to $\mathcal{A}_{\mathtt{AE}}$. $\mathcal{A}_{\mathtt{AE}}$ sets $q_0 = q^*$ and selects a random q_1. Then flipping a bit b', it sends $\langle 0, q_{b'}\rangle$ to the AE challenger. The challenger selects a random key $\tilde{K}$ and flips a bit b. If $b = 0$, it runs $AE.Enc(\tilde{K}, 0, q_{b'})$ and returns the result as C^*. Else, a random C^* is returned. $\mathcal{A}_{\mathtt{AE}}$ runs $KEM.Enc(pk)$ to obtain $\langle K^*, C_0^*\rangle$, discards K^* and returns $A^* = C_0^*||C^*$ to $\mathcal{A}$ as the challenge.

Further queries for $OQEM.Dec(A = C_0||C)$ by $\mathcal{A}$ are answered similarly as before, except for $C_0 = C_0^*$, where $\langle 0, C\rangle$ is sent to $AE.Dec(\cdot)$. (Note that C must differ from C^*). Queries for $OIREM.Enc(i, r)$ are answered by $\mathcal{A}_{\mathtt{AE}}$ who sets $r_0 = r$ and selects a random r_1 and gives $\langle i, r_{b'}\rangle$ to the AE challenger, where b' is that chosen by $\mathcal{A}_{\mathtt{AE}}$ during the challenge phase. AE challenger returns $B = AE.Enc(\tilde{K}, i, r_{b'})$ if b chosen by it during the previous challenge phase is 0, else it returns a random string. When $\mathcal{A}$ outputs a bit $\tilde{b}$, $\mathcal{A}_{\mathtt{AE}}$ outputs $\tilde{b} \oplus b'$.

For $b = 0$, $\mathcal{A}$ plays the IND-CCA-QREM* game with $\mathcal{A}_{\mathtt{AE}}$. Further, let X^{**} be the event that $\mathcal{A}$ wins conditioned to $b = 1$. We obtain

$$\begin{aligned}\mathtt{Pr}[\mathcal{A}_{\mathtt{AE}}\ \mathtt{wins}] - \frac{1}{2} &= \mathtt{Pr}[\tilde{b} \oplus b' = b] - \frac{1}{2} = \frac{1}{2}(\mathtt{Pr}[\tilde{b} = b'|b = 1] - \mathtt{Pr}[\tilde{b} = b'|b = 0]) \\ &= \frac{1}{2}(\mathtt{Pr}[X^{**}] - \mathtt{Pr}[X^*]).\end{aligned}$$

Since $|\mathtt{Pr}[\mathcal{A}_{\mathtt{AE}}\ \mathtt{wins}] - \frac{1}{2}| \leq Adv_{\pi_{\mathtt{AE}}}^{\mathtt{AE}}(\lambda)$, thus $|\mathtt{Pr}[X^{**}] - \mathtt{Pr}[X^*]| \leq 2Adv_{\pi_{\mathtt{AE}}}^{\mathtt{AE}}(\lambda)$. Also note that for $b = 1$ no information on b' leaks, so $\mathtt{Pr}[X^{**}] = \frac{1}{2}$. Rearranging, we have

$$\begin{aligned}\mathtt{Pr}[X] - \frac{1}{2} &\leq 2Adv_{\pi_{\mathtt{KEM}}}^{\mathtt{IND-CCA}}(\lambda) + 2Adv_{\pi_{\mathtt{AE}}}^{\mathtt{AE}}(\lambda) \\ Adv_{\pi_{\mathtt{QREM}}}^{\mathtt{IND-CCA-QREM}}(\lambda) &\leq 2Adv_{\pi_{\mathtt{KEM}}}^{\mathtt{IND-CCA}}(\lambda) + 2Adv_{\pi_{\mathtt{AE}}}^{\mathtt{AE}}(\lambda).\end{aligned}$$

Authentication. We consider reducing the AUTH-CCA-QREM game. This is similar to the previous reduction for IND-CCA-QREM, so we will only highlight the differences. The simulation proceeds similarly, except during the challenge phase. In more detail, the IND-CCA-KEM challenger runs $KEM.Enc(pk)$ to obtain $\langle K^\dagger, C_0^*\rangle$. It then flips a bit b and if $b = 0$ it returns $\langle K^* = K^\dagger, C_0^*\rangle$ to $\mathcal{A}_{\mathtt{KEM}}$. Else it returns a random key $\tilde{K}$ as K^*. When $\mathcal{A}_{\mathtt{KEM}}$ receives q^* from $\mathcal{A}$ it computes $C^* = AE.Enc(K^*, 0, q^*)$ and returns $A^* = C_0^*||C^*$ as the challenge.

Queries for $OREM.Enc(i, r^*)$ can be easily answered by $\mathcal{A}_{\mathtt{KEM}}$ since it knows K^*, i.e. it returns $B^* = AE.Enc(K^*, i, r^*)$; additionally each of these B^* are recorded.

Finally, when $\mathcal{A}$ outputs a $\langle i, B\rangle$, $\mathcal{A}_{\mathtt{KEM}}$ runs $AE.Dec(K^*, i, B)$ and checks the resulting r. If no $\perp$ is obtained (meaning B is valid) and if B does not match with a previously recorded B^*, then it is clear that $\mathcal{A}$ wins, and thus $\mathcal{A}_{\mathtt{KEM}}$ outputs a guess $\tilde{b} = 0$. Else it outputs $\tilde{b} = 1$.

When $b = 0$, $\mathcal{A}$ plays the AUTH-CCA-QREM game with $\mathcal{A}_{\mathtt{KEM}}$. Let AUTH-CCA-QREM* be the modified game corresponding to $b = 1$, and X and X^* be the corresponding events that $\mathcal{A}$ wins in these AUTH-CCA-QREM and AUTH-CCA-QREM* games respectively:

$$\begin{aligned}\mathtt{Pr}[\mathcal{A}_{\mathtt{KEM}}\ \mathtt{wins}] - \frac{1}{2} &= \mathtt{Pr}[\tilde{b} = b] - \frac{1}{2} = \frac{1}{2}(\mathtt{Pr}[\tilde{b} = 0|b = 1] - \mathtt{Pr}[\tilde{b} = 0|b = 0]) \\ &= \frac{1}{2}(\mathtt{Pr}[\mathcal{A}\ \mathtt{wins}|b = 1] - \mathtt{Pr}[\mathcal{A}\ \mathtt{wins}|b = 0]) = \frac{1}{2}(\mathtt{Pr}[X^*] - \mathtt{Pr}[X]).\end{aligned}$$

Since $|\mathtt{Pr}[\mathcal{A}_{\mathtt{KEM}}\ \mathtt{wins}] - \frac{1}{2}| \leq Adv^{\mathtt{IND-CCA}}_{\pi_{\mathtt{KEM}}}(\lambda)$, thus $|\mathtt{Pr}[X^*] - \mathtt{Pr}[X]| \leq 2Adv^{\mathtt{IND-CCA}}_{\pi_{\mathtt{KEM}}}(\lambda)$.

Consider now reducing AUTH-CCA-QREM* to the AE game. The steps are similar to the case for IND-CCA-QREM* so we only mention the differences.

During the challenge stage, when $\mathcal{A}_{\mathtt{AE}}$ receives q^* from $\mathcal{A}$ it forwards $\langle 0, q^* \rangle$ to the AE challenger, who selects a random key $\tilde{K}$ and flips a bit b. If $b = 0$, it runs $AE.Enc(\tilde{K}, 0, q^*)$ and returns the result as C^*. Else, a random C^* is returned. $\mathcal{A}_{\mathtt{AE}}$ runs $KEM.Enc(pk)$ to obtain $\langle K^*, C^*_0 \rangle$, discards K^* and returns $A^* = C^*_0 || C^*$ to $\mathcal{A}$ as the challenge.

Finally, when $\mathcal{A}$ outputs a B, $\mathcal{A}_{\mathtt{AE}}$ checks that B does not equal the output of any previous $OREM.Enc(i, r)$ query, and passes r to the AE challenger who depending on the bit b selected during the challenge phase either returns the result $r = AE.Dec(\tilde{K}, i, B)$ or $\perp$. If no $\perp$ is obtained meaning $\mathcal{A}$ wins, then $\mathcal{A}_{\mathtt{AE}}$ outputs a guess $\tilde{b} = 0$. Else it outputs $\tilde{b} = 1$.

Let X^{**} be the event that $\mathcal{A}$ wins the game conditioned to $b = 1$. We obtain

$$\begin{aligned}\mathtt{Pr}[\mathcal{A}_{\mathtt{AE}}\ \mathtt{wins}] - \frac{1}{2} &= \mathtt{Pr}[\tilde{b} = b] - \frac{1}{2} = \frac{1}{2}(\mathtt{Pr}[\tilde{b} = 0|b = 1] - \mathtt{Pr}[\tilde{b} = 0|b = 0]) \\ &= \frac{1}{2}(\mathtt{Pr}[\mathcal{A}\ \mathtt{wins}|b = 1] - \mathtt{Pr}[\mathcal{A}\ \mathtt{wins}|b = 0]) = \frac{1}{2}(\mathtt{Pr}[X^{**}] - \mathtt{Pr}[X^*]).\end{aligned}$$

Since $|\mathtt{Pr}[\mathcal{A}_{\mathtt{AE}}\ \mathtt{wins}] - \frac{1}{2}| \leq Adv^{\mathtt{AE}}_{\pi_{\mathtt{AE}}}(\lambda)$, and $\mathtt{Pr}[X^{**}] = 0$; thus $|\mathtt{Pr}[X^{**}] - \mathtt{Pr}[X^*]| \leq 2Adv^{\mathtt{AE}}_{\pi_{\mathtt{AE}}}(\lambda)$.

Rearranging, we have

$$\begin{aligned}\mathtt{Pr}[X] &\leq 2Adv^{\mathtt{IND-CCA}}_{\pi_{\mathtt{KEM}}}(\lambda) + 2Adv^{\mathtt{AE}}_{\pi_{\mathtt{AE}}}(\lambda) \\ Adv^{\mathtt{AUTH-CCA-QREM}}_{\pi_{\mathtt{QREM}}}(\lambda) &\leq 2Adv^{\mathtt{IND-CCA}}_{\pi_{\mathtt{KEM}}}(\lambda) + 2Adv^{\mathtt{AE}}_{\pi_{\mathtt{AE}}}(\lambda).\end{aligned}$$

□

B Proof of Theorem 2

Proof. Let Q be the number of protocol sessions initiated by $\mathcal{S}$ in the encapsulate-system; and $l + 1$ be the number of messages within each protocol session. We take an adversary $\mathcal{E}$ against the encapsulate-system $\mathcal{S}^{enc} \leftrightarrow \mathcal{O}$ of complexity $c - \mu$ where $\mu = \mathtt{max}(\mu_1, \mu_2, \mu_3)$ with μ_1, μ_2, μ_3 to be later defined. We construct a sequence of hybrid systems $\mathcal{S}_i$ and $\mathcal{S}'_i$ for $i = 1, \ldots, Q$ with $\mathcal{S}_0 = \mathcal{S}^{enc}$, and $\mathcal{S}'_0 = \mathcal{S}_Q$ interacting with the adversary $\mathcal{E}$ and oracle $\mathcal{O}$. $\mathcal{S}'_Q$ will almost be the toll-system $\mathcal{S}^{toll}$.

$\mathcal{S}_i$ and $\mathcal{S}'_i$ keep record of all $\langle A', B'_1, B'_2, B'_3, \ldots, B'_l, q', r'_1, r'_2, r'_3, \ldots, r'_l \rangle$. System $\mathcal{S}'_i$ further keeps additional record of some $\langle K, A, q \rangle$ triplets.

We define the oracle-system $\mathcal{S}_i \leftrightarrow \mathcal{O}$ which slightly differs from $\mathcal{S}_{i-1} \leftrightarrow \mathcal{O}$. Let A be the ith encapsulated query from $\mathcal{S}$, i.e. the value of the ith "QUERY A" message. It encapsulates the ith query q. Let B_t (for $t = 1, 2, \ldots, l$) be the encapsulations of the corresponding responses r_t to this ith query q. If the environment submits a "RESPOND B_t" message ($t \in \{1, 2, \ldots, l\}$) to $\mathcal{S}_i$, the system checks if some $\langle A', B'_1, B'_2, B'_3, \ldots, q', r'_1, r'_2, r'_3, \ldots \rangle$ with $A' = A$ and $B'_s = B_s$ (for $s \leq t$) exists in its record. r'_t is returned if this is so. In other cases, S_i aborts.

$\mathcal{S}_{i-1}$ and $\mathcal{S}_i$ only differ in the treatment on the "RESPOND B_t" (for $t = 1, 2, \ldots, l$) messages initiated by the ith "QUERY A" message. They can be simulated without the initial secret key sk provided that we get pk and we can use an $OQEM.Dec(\cdot)$ oracle to treat "QUERY A'" with $A' \neq A$ messages. We define an adversary $\mathcal{A}_i$ against the AUTH-CCA-QREM game this way (See Fig. 4): The adversary first receives the public key pk and simulates the interaction with adversary $\mathcal{E}$ and $\mathcal{O}$ until $\mathcal{S}$ issues its ith query q. Then, $\mathcal{A}_i$ submits q^* to the challenger and receives an encapsulation A^*. The adversary can call the $OQEM.Dec(\cdot)$ oracle except for input A^* to treat the "QUERY A'" message with $A' \neq A^*$. To treat the "QUERY A'" and subsequent "RESPOND B'_t" messages with $A' = A^*$ and $B'_s = B_s$ (for $s \leq t$), recall that $\mathcal{A}_i$ knows q^* so it simply queries $\mathcal{O}$ with q^* as many times as necessary to get r^*_{t+1} and calls $OREM.Enc(t+1, r^*_{t+1})$ to get B^*_{t+1}. Note that it is important to query $\mathcal{O}$ every time since the oracle may not be deterministic, e.g. stateful oracles. The final B_l in the "RESPOND B_l" message to the system is the final B_l in the AUTH-CCA-QREM game, so the simulation is perfect. In the event E that B_l equals one of the obtained B^*_t from $OREM.Enc$, then $\mathcal{A}_i$ fails to win the AUTH-CCA-QREM game; in which case the behavior of $\mathcal{E}(\mathcal{S}_{i-1}, \mathcal{O})$ and $\mathcal{E}(\mathcal{S}_i, \mathcal{O})$ are identical, and $\mathcal{A}_i$ perfectly simulates $\mathcal{S}_{i-1}$ or $\mathcal{S}_i$ to $\mathcal{E}$. Hence $\Pr[\mathcal{E}(\mathcal{S}_{i-1}, \mathcal{O})\ \texttt{wins}|E] = \Pr[\mathcal{E}(\mathcal{S}_i, \mathcal{O})\ \texttt{wins}|E]$. Otherwise, $\mathcal{A}_i$ wins. If Q/REM is AUTH-CCA secure, the advantage for $\mathcal{A}_i$ is negligible, meaning that $\Pr[\mathcal{A}_i\ \texttt{wins}] \leq \varepsilon_{auth}$. We deduce $|\Pr[\mathcal{E}(\mathcal{S}_{i-1}, \mathcal{O})\ \texttt{wins}] - \Pr[\mathcal{E}(\mathcal{S}_i, \mathcal{O})\ \texttt{wins}]| \leq \varepsilon_{auth}$ since the complexity of $\mathcal{A}_i$ is $c - \mu + \mu_1$ for some small overhead cost μ_1.

We define the oracle-system $\mathcal{S}'_i \leftrightarrow \mathcal{O}$ which slightly differs from $\mathcal{S}'_{i-1} \leftrightarrow \mathcal{O}$. Now, instead of encapsulating q to produce $\langle K, A\rangle$ in the ith query from $\mathcal{S}$, $\mathcal{S}'_i$ encapsulates a random query $\tilde{q}$ of the same length and keeps record of $\langle K, A, q\rangle$. Similarly, for any "QUERY A'" message from the environment with $A' = A$ for some $\langle K, A, q\rangle$ record, $\mathcal{S}'_i$ gets q from the record, queries $\mathcal{O}$ with $q' = q$, and obtains the response $r'_1 = r_1$. Nevertheless, instead of encapsulating r_1, the system directly encapsulates a random response $\tilde{r}_1$ to produce B'_1. The record $\langle A', B'_1, q, r_1\rangle$ is inserted. This querying of the oracle $\mathcal{O}$ is just to update the internal state of $\mathcal{O}$ to handle the case of stateful oracles. The same occurs when producing subsequent response messages r_{t+1} ($t \in \{1, 2, \ldots, l\}$) triggered by a preceeding message r_t i.e. a random $\tilde{r}_{t+1}$ is encapsulated instead of querying $\mathcal{O}(r_t)$ and $\langle B'_{t+1}, r_{t+1}\rangle$ are added to the record.

$\mathcal{S}'_{i-1}$ and $\mathcal{S}'_i$ only differ in the treatment on the ith "QUERY A" message and related "QUERY A'" messages with $A' = A$; and corresponding "RESPOND B_t" and "RESPOND B'_t" messages. They can be simulated without the initial secret key sk provided that we get pk and we can use an $OQEM.Dec(\cdot)$ oracle to treat "QUERY A'" with $A' \neq A$ messages; and $OREM.Dec(\cdot)$ oracle to treat "RESPOND B_t" with $B'_s \neq B_s (s \leq t)$ messages. We define an adversary $\mathcal{A}'_i$ against the IND-CCA-QREM game this way (See Fig. 5): The adversary first receives the public key pk and simulates the interaction with adversary $\mathcal{E}$ and $\mathcal{O}$ until $\mathcal{S}$ issues its ith query q^*. Then, $\mathcal{A}'_i$ submits q^* to the challenger and receives an encapsulation A^*. The adversary can call the $OQEM.Dec(\cdot)$ oracle except for input A^* to treat the "QUERY A'" message with $A' \neq A^*$. To treat the "QUERY A'" message with $A' = A^*$, $\mathcal{A}'_i$ knows q^* so it simply queries $\mathcal{O}$ with q^* as many times as necessary to get r^*_t (for $t = 1, 2, \ldots, l$). and calls $OIREM.Enc(t, r^*)$ to get B^*. The success of the Γ-adversary against the system yields the final guess bit in the IND-CCA-QREM game. If Q/REM is IND-CCA secure, the advantage for $\mathcal{A}'_i$ is negligible, meaning that $\Pr[\mathcal{A}'_i\ \texttt{wins}] \leq \varepsilon_{sem}$. We deduce $|\Pr[\mathcal{E}(\mathcal{S}'_{i-1}, \mathcal{O})\ \texttt{wins}] - \Pr[\mathcal{E}(\mathcal{S}'_i, \mathcal{O})\ \texttt{wins}]| \leq \varepsilon_{sem}$ since the complexity of $\mathcal{A}'_i$ is $c - \mu + \mu_2$ for some small overhead cost μ_2.

$\mathcal{S}'_Q$ is such that whenever $\mathcal{E}(\mathcal{S}'_Q, \mathcal{O})$ wins for any i, between any ith QUERY A and ith "RESPOND B_t" messages, there is at least one "QUERY A'" with $A' = A$ and $B'_s = B_s (s \leq t)$ from $\mathcal{E}$ (otherwise some modified treatment of the "RESPOND B_t" message brought by $\mathcal{S}_i$ fails, since if $A' = A$ and and $B'_s = B_s (s \leq t)$ then $\mathcal{S}_i$ will not answer r'_t to $\mathcal{S}$). Due the assumption on $\mathcal{O}$, discarding repeating queries is harmless. We define a new system $\mathcal{S}'' \leftrightarrow \mathcal{O}$ which differs from $\mathcal{S}'_Q \leftrightarrow \mathcal{O}$ in the sense that for any "QUERY A'" or "RESPOND B'_t"

message such that there is one record $\langle K, A, B_1, B_2, \ldots, q, r_1, r_2, \ldots \rangle$ with $A' = A$ then $\mathcal{O}$ is not queried but $\mathcal{S}''$ picks a random $\tilde{r}$ of same length as r', does $REM.Enc(K^*, t, \tilde{r}) = B'_t$ and answers "RESPOND B'_t". Obviously, $\Pr[\mathcal{E}(S'', \mathcal{O})\ \texttt{wins}] \geq \Pr[\mathcal{E}(S'_Q, \mathcal{O})\ \texttt{wins}]$.

Finally we construct a new adversary $\mathcal{E}'$ against $\mathcal{S}^{toll}$ from the adversary $\mathcal{E}$ attacking $\mathcal{S}''$. See Fig. 6. $\mathcal{E}'$ first generates $QEM.Key(1^\lambda) = \langle pk, sk \rangle$ and simulates $\mathcal{E}$ with input pk. Interactions between $\mathcal{E}$ and $\mathcal{S}$ remain unchanged. When $\mathcal{E}'$ receives a "QUERY l" message from $\mathcal{S}^{toll}$, $\mathcal{E}'$ picks a random $\tilde{q}$ of length l, runs $QEM.Enc(pk, \tilde{q}) = \langle K', A' \rangle$ and sends a "QUERY A" message to $\mathcal{E}$. When $\mathcal{E}$ yields a "QUERY A'" message with $A' = A$, a "GO" message is given to $\mathcal{S}^{toll}$ and a random response $\tilde{r}_1$ is encapsulated. When $\mathcal{S}^{toll}$ issues a "RESPOND m" message to $\mathcal{E}'$, then $\mathcal{E}'$ answers "RESPOND B'_t" to $\mathcal{E}$. Finally when $\mathcal{E}$ sends a "RESPOND B'_t" message to $\mathcal{E}'$, then $\mathcal{E}'$ sends a "GO" message to $\mathcal{S}^{toll}$. Obviously, $\Pr[\mathcal{E}'(S^{toll}, \mathcal{O})\ \texttt{wins}] = \Pr[\mathcal{E}(S'', \mathcal{O})\ \texttt{wins}]$. Since $\mathcal{S}$ is ε-$\mathit{\Gamma}$-toll-secure with $\mathcal{O}$, we know that $\Pr[\mathcal{E}(S^{toll}, \mathcal{O})\ \texttt{wins}] \leq \varepsilon$ since the complexity of $\mathcal{E}'$ is $c - \mu + \mu_3$ for some small overhead cost μ_3. Hence, $\Pr[\mathcal{E}(S^{enc}, \mathcal{O})\ \texttt{wins}] \leq (Q(\varepsilon_{sem} + \varepsilon_{auth}) + \varepsilon)$.

□

$\mathcal{E}$		$\mathcal{A}_i$ simulates S_i		QREM Challenger
	$\xleftarrow{pk}$		$\xleftarrow{pk}$	$QEM.Key(1^\lambda) = \langle pk, sk\rangle$.
	$\xleftarrow{\texttt{QUERY}\ A}$	`Case:` Receive jth query q (for $j \neq i$) from $\mathcal{S}$. $QEM.Enc(pk, q) = \langle K, A\rangle$. Keep $\langle j, K\rangle$ in memory.		
	$\xrightarrow{\texttt{QUERY}\ A'}$ $\xleftarrow{\texttt{RESPOND}\ B'_1}$	If $q' = \perp$ then halt. Else $\mathcal{O}(q') = r'_1$. $REM.Enc(K', 1, r'_1) = B'_1$. Record $\langle K', A', B'_1, q', r'_1\rangle$.	$\xrightarrow{OQEM.Dec(A')}$ $\xleftarrow{\langle K', q'\rangle}$.	$QEM.Dec(sk, A') = \langle K', q'\rangle$.
	$\xrightarrow{\texttt{RESPOND}\ B_t(t=1,\dots)}$ $\xleftarrow{\texttt{RESPOND}\ B'_{t+1}}$	For $(j < i)$, if $\langle K', A', B'_1, B'_2, \dots, q', r'_1, r'_2, \dots,\rangle$ exists s.t. $A' = A$ and $B'_s = B_s (s \leq t)$, answer r'_t to $\mathcal{S}/\mathcal{O}$ else halt. For $(j > i)$, $REM.Dec(K, t, B_t) = r_t$. If $r_t = \perp$ then halt. Else answer r_t to $\mathcal{S}/\mathcal{O}$ and get r'_{t+1}. $REM.Enc(K', t+1, r'_{t+1}) = B'_{t+1}$. Add $\langle B'_{t+1}, r'_{t+1}\rangle$ to the record.		
	$\xleftarrow{\texttt{QUERY}\ A^*}$	`Case:` Receive ith query q^* from S. Keep $\langle q^*, A^*\rangle$ in memory.	$\xrightarrow{q^*}$ $\xleftarrow{A^*}$	$QEM.Enc(pk, q^*) = \langle K^*, A^*\rangle$.
	$\xrightarrow{\texttt{QUERY}\ A'}$ $\xleftarrow{\texttt{RESPOND}\ B'_1}$	If $A' \neq A^*$, then proceed as for normal QUERY A' $\vdots$	$\vdots$	$\vdots$
	$\xrightarrow{\texttt{QUERY}\ A'}$ $\xleftarrow{\texttt{RESPOND}\ B^*_1}$	If $A' = A^*$, then $\mathcal{O}(q^*) = r^*_1$. Record $\langle A^*, B^*_1, q^*, r^*_1\rangle$.	$\xrightarrow{OREM.Enc(1, r^*_1)}$ $\xleftarrow{B^*_1}$	$REM.Enc(K^*, 1, r^*_1) = B^*_1$.
	$\xrightarrow{\texttt{RESPOND}\ B'_t(t=1,\dots)}$ $\xleftarrow{\texttt{RESPOND}\ B^*_{t+1}}$	If $\langle A', B'_1, B'_2, \dots, q', r'_1, r'_2, \dots\rangle$ exists s.t. $A' = A^*$ and $B'_t = B^*_t$, then answer r'_t to $\mathcal{S}/\mathcal{O}$ else halt. Get r^*_{t+1}. Add $\langle B^*_{t+1}, r^*_{t+1}\rangle$ to the record.	$\xrightarrow{OREM.Enc(t+1, r^*_{t+1})}$ $\xleftarrow{B^*_{t+1}}$	$REM.Enc(K^*, t+1, r^*_{t+1}) = B^*_{t+1}$.
	$\xrightarrow{\texttt{RESPOND}\ B_l}$	If $\langle A', B'_1, B'_2, \dots, B'_l, q', r'_1, r'_2, \dots, r'_l\rangle$ exists s.t. $A' = A^*$ and $B'_l = B_l$, answer r'_l to $\mathcal{S}$. Else, output B_l to QREM challenger.	$\xrightarrow{B_l}$	

Fig. 4. Reduction from Γ-encapsulate-secure* to AUTH-CCA-QREM

$\mathcal{E}$		$\mathcal{A}_i'$ simulates S_i'		QREM Challenger
	$\xleftarrow{pk}$		$\xleftarrow{pk}$	$QEM.Key(1^\lambda) = \langle pk, sk\rangle$.
	$\xleftarrow{\texttt{QUERY } A}$	Receive jth query q from $\mathcal{S}$. If $(j < i)$: Set $q^\dagger = \tilde{q}$ for random $\tilde{q}$. $QEM.Enc(pk, q^\dagger) = \langle K, A\rangle$. Record $\langle K, A, q\rangle$. If $(j > i)$: Set $q^\dagger = q$. $QEM.Enc(pk, q^\dagger) = \langle K, A\rangle$. If $(j = i)$: $\vdots$ Record $\langle \perp, A^*, q\rangle$.	$\xrightarrow{q^*=q}$ $\xleftarrow{A^*}$	Choose $b \in \{0, 1\}$. Set $q_0 = q^*$. Select random q_1. $QEM.Enc(pk, q_b) = \langle K^*, A^*\rangle$.
	$\xrightarrow{\texttt{QUERY } A'}$ $\xleftarrow{\texttt{RESPOND } B_1'}$	If $\langle K, A, q\rangle$ exists s.t. $A' = A$, then: $\mathcal{O}(q) = r_1$. If $K \neq \perp$ then set $r^\dagger = \tilde{r}$ for random $\tilde{r}$. $REM.Enc(K, 1, r^\dagger) = B_1'$. Else if $K = \perp$ call $OIREM.Enc(1, r_1)$. Set $B_1' = B^*$. Record $\langle A', B_1', q, r_1\rangle$. Else call $OQEM.Dec(A')$. If $q' = \perp$ then halt. $\mathcal{O}(q') = r_1'$. $REM.Enc(K', 1, r_1') = B_1'$. Record $\langle A', B_1', q', r_1'\rangle$.	$\xrightarrow{OIREM.Enc(1,r_1)}$ $\xleftarrow{B^*}$ $\xrightarrow{OQEM.Dec(A')}$ $\xleftarrow{\langle K', q'\rangle}$	Set $r_0 = r_1$. Select random r_1. $REM.Enc(K^*, 1, r_b) = B^*$. $QEM.Dec(sk, A') = \langle K', q'\rangle$.
	$\xrightarrow{\texttt{RESPOND } B_t}$ $\xleftarrow{\texttt{RESPOND } B_{t+1}'}$	If $\langle K, A, B_1, B_2 \ldots, q, r_1, r_2, \ldots\rangle$ exists s.t. $A' = A$ and $B_s' = B_s (s \leq t)$, then: Answer r_t to $\mathcal{S}/\mathcal{O}$ and get r_{t+1}. If $K \neq \perp$ then set $r^\dagger = \tilde{r}$ for random $\tilde{r}$. $REM.Enc(K, t+1, r^\dagger) = B_{t+1}'$. Else if $K = \perp$ call $OIREM.Enc(t+1, r_{t+1})$. Set $B_{t+1}' = B^*$. Add $\langle B_{t+1}', r_{t+1}\rangle$ to record. Else call $OREM.Dec(t+1, B_{t+1}')$. If $r_{t+1}' = \perp$ then halt. Add $\langle B_{t+1}', r_{t+1}'\rangle$ to record.	$\xrightarrow{OIREM.Enc(t+1,r_{t+1})}$ $\xleftarrow{B^*}$ $\xrightarrow{OREM.Dec(t+1,B_{t+1}')}$ $\xleftarrow{r_{t+1}'}$	Set $r_0 = r_{t+1}$. Select random r_1. $REM.Enc(K^*, t+1, r_b) = B^*$. $REM.Dec(K^*, t+1, B_{t+1}') = r_{t+1}'$.
	$\xrightarrow{\texttt{RESPOND } B_l}$	If $\langle A', B_1', \ldots, B_l', q', r_1', \ldots, r_l'\rangle$ exists s.t. $A' = A$ and $B_l' = B_l$, answer r_l' to $\mathcal{S}$.		
	$\xrightarrow{\Gamma-\texttt{Judge: end}}$	If $\mathcal{A}$ wins, output $\tilde{b} = 1$, else $\tilde{b} = 0$.	$\xrightarrow{\tilde{b}}$	

Fig. 5. Reduction from Γ-encapsulate-secure** to IND-CCA-QREM

$\mathcal{E}$		$\mathcal{A}'$		$\mathcal{S}^{toll}$
	$\xleftarrow{pk}$	$QEM.Key(1^\lambda) = \langle pk, sk\rangle$.		
	$\xrightarrow{\texttt{QUERY } A'}$ $\xleftarrow{\texttt{RESPOND } B_1'}$	$QEM.Dec(sk, A') = \langle K', q'\rangle$. $\mathcal{O}(q') = r_1'$. $REM.Enc(K', 1, r_1') = B_1'$.		
	$\xrightarrow{\texttt{RESPOND } B_t}$ $\xleftarrow{\texttt{RESPOND } B_{t+1}'}$	$REM.Dec(K', t, B_t) = r_t'$. $\mathcal{O}(r_t') = r_{t+1}'$. $REM.Enc(K', t+1, r_{t+1}') = B_{t+1}'$.		
	$\xleftarrow{\texttt{QUERY } A}$ $\xrightarrow{\texttt{QUERY } A'}$ $\xleftarrow{\texttt{RESPOND } B_1'}$	Pick random $\tilde{q}$ of length l. $QEM.Enc(pk, \tilde{q}) = \langle K, A\rangle$. If $A' \neq A$, do as before. Else if $A' = A$, pick random $\tilde{r}$ of length m. $REM.Enc(K, 1, \tilde{r}) = B_1'$.	$\xleftarrow{\texttt{QUERY } l}$ $\xrightarrow{\texttt{GO}}$ $\xleftarrow{\texttt{RESPOND } m}$	
	$\xrightarrow{\texttt{RESPOND } B_t}$	If $B_t' \neq B_t$, do as before. Else pick random $\tilde{r}$ of length m. $REM.Enc(K, t+1, \tilde{r}) = B_{t+1}'$.	$\xrightarrow{\texttt{GO}}$	

Fig. 6. Going from $\mathcal{S}''$ to $\mathcal{S}^{toll}$