entropy

MDPI

Article

From Key Encapsulation to Authenticated Group Key Establishment—A Compiler for Post-Quantum Primitives [†]

Edoardo Persichetti [1,‡], Rainer Steinwandt [1,‡] and Adriana Suárez Corona [2,*,‡]

1 Department of Mathematical Sciences, Florida Atlantic University, Boca Raton, FL 33431, USA; epersichetti@fau.edu (E.P.); rsteinwa@fau.edu (R.S.)
2 Department of Mathematical Sciences, Universidad de León, 24071 León, Spain
* Correspondence: asuac@unileon.es

† This paper is an extended version of our paper published in JNIC 2019 (as 2-page extended abstract and as poster) held at the San Francisco de Cáceres Complex between June 5 and 7, 2019.
‡ These authors contributed equally to this work.

Received: 30 September 2019; Accepted: 27 November 2019; Published: 30 November 2019

Abstract: Assuming the availability of an existentially unforgeable signature scheme and an (IND- CCA secure) key encapsulation mechanism, we present a generic construction for group key establishment. The construction is designed with existing proposals for post-quantum cryptography in mind. Applied with such existing proposals and assuming their security, we obtain a quantum-safe three-round protocol for authenticated group key establishment that requires only one signature per protocol participant.

Keywords: authenticated group key establishment; post-quantum cryptography; key encapsulation mechanism

1. Introduction

To enable confidential communication among a group of two or more users over an insecure network, cryptography provides group key establishment procotols. Dealing with a non-trusted communication infrastructure, the question of authenticating protocol participants naturally arises in this context as well. The resulting cryptographic protocols are commonly application-agnostic and focus only on the task of establishing a shared high-entropy secret among the legitimate users, not mandating any particular usage of that secret key in subsequent applications. The resulting task, authenticated group key establishment (AGKE), is fairly well-understood, even though there is a remarkable diversity in the details of security models in use. A standard technique to derive an AGKE solution is to apply some form of protocol compiler or generic framework to a passively secure solution. If a public-key infrastructure is available, signatures provide an adequate mechanism. This results commonly in protocols with a substantial number of signatures being computed, transmitted, and verified:

- In the Katz-Yung compiler [1], for each message sent in the original protocol, a signature has to be computed and transmitted (and verified). For instance, Apon et al. [2] propose the application of this compiler for their unauthenticated group key establishment solution.
- The compiler made by Bresson et al., `C-AMA` [3] requires a signature for each message in the original protocol, plus one more for each protocol participant (and according signature verifications).
- Bohli's framework for robust group key agreement [4] targets two-round protocols, and in each round each participant sends a signed message (and verifies signatures by all other participants).

A popular AGKE building block is the traditional Diffie–Hellman two-party key exchange. This one-round protocol for two parties enables elegant two-round solutions for group key establishment—the Burmester–Desmedt protocol [5] offering a prominent design. Using a compiler (or a tailored design approach), deriving an AGKE solution with two or three rounds has by now become standard practice. Regrettably, owing to Shor's algorithm for solving discrete logarithms [6], the traditional Diffie–Hellman protocol is no longer a viable building block for post-quantum AGKE. While the cryptanalytic threat of quantum computers should not be overstated (cf., e.g., [7,8]), in view of steady technological progress, it is, therefore, necessary to provide new solutions that will be robust in the post-quantum setting.

With the current status of quantum cryptanalysis, it is clear that mathematical tools relying on the hardness of factoring or computing discrete logarithms are no longer an option. Regrettably, no "silver bullet" is known that would allow a seamless replacement of today's solutions with quantum-safe ones. In this scenario, the cost of integrating signatures in a protocol can differ quite a bit from the familiar setting. Specifically, for hash-based designs, arguably one of the most popular approaches for post-quantum signing, the size of signatures remains an issue. For instance, proposed instances of the SPHINCS$^+$ design have signature lengths between 8080 and 49,216 bytes [9]. Taking this into account, a compiler by Tang and Mitchell [10] appears more attractive from today's post-quantum perspective. Assuming the a priori availability of a unique session identifier that is distributed among the protocol participants, Tang and Mitchell present a compiler where each participant computes and transmits only one signature (and performs signature verifications).

Our Contribution

With the current state of the the art, key encapsulation mechanisms (KEMs) are a natural starting point for implementing key establishment solutions in a post-quantum setting. This is evident, for example, when looking at the ongoing NIST standardization effort [11], where the vast majority of candidates for public-key encryption and key exchange are indeed presented as KEMs. In this paper, we show how to derive a three-round AGKE using KEMs, where no participant signs more than one message. This is one round less than applying the Katz–Yung compiler to Apon et al.'s unauthenticated three-round protocol [2], and differing from the Katz–Yung-based construction, our approach requires only one signature per protocol participant. The signature scheme we use is assumed to be EUF-CMA secure, and the KEM we involve is assumed to be IND-CCA secure.

2. Preliminaries

Here and in the subsequent sections, the security parameter will be denoted by ℓ, and notions like polynomial time or negligible refer to that parameter.

2.1. Key Encapsulation Mechanism

A main technical tool in our construction are key encapsulation mechanisms (introduced by Shoup in [12]), which are a structure similar to public-key encryption schemes, with the main difference being that the goal is to encrypt ("encapsulate") a randomly-generated symmetric key. A formal definition is as follows.

Definition 1 (Key Encapsulation Mechanism)**.** *A key encapsulation mechanism (KEM) is a triple of polynomial time algorithms* (`KeyGen`, `Encaps`, `Decaps`) *as follows:*

`KeyGen` *is probabilistic. Given the security parameter ℓ, it generates a pair of public and secret keys (pk, sk).*

`Encaps` *is probabilistic. Given a public key pk, it generates a pair (K, C) where $K \in \{0,1\}^\ell$ is a symmetric key and C is an encapsulation of this key under the public key pk.*

`Decaps` *is deterministic. Given a secret key sk and an encapsulation C, this algorithm outputs the symmetric key K or a special error symbol $\perp$.*

For our purposes, the KEM needs to be perfectly correct, i.e., we require that for all key pairs (pk, sk) generated by `KeyGen` the following correctness condition holds: if (K, C) is an output of $\texttt{Encaps}(pk)$, then $\texttt{Decaps}_{sk}(C) = K$.

The most relevant notion to capture the security of a KEM is that of indistinguishability under chosen-ciphertext attacks (also defined in [12]), which is once again similar to the usual one for PKEs, except that the adversary is now asked to distinguish between an honestly encapsulated key and a pseudorandom value. We provide a formal definition below.

Definition 2 (IND-CCA Security). *A KEM is IND-CCA secure if the advantage of any probabilistic polynomial time adversary $\mathcal{A}$ in the game described in Figure 1, as a function in the security parameter, is negligible. Here the advantage of an adversary $\mathcal{A}$ is defined as* $\mathrm{Adv}_{\mathcal{A}}^{\text{IND-CCA}}(\ell) = |2 \cdot \Pr[b = b'] - 1|$.

Setup: the challenger $\mathcal{C}$ runs the key generation `KeyGen` to obtain a key pair (pk, sk) and hands the public key pk to the adversary $\mathcal{A}$.
Queries: the adversary $\mathcal{A}$ is given access to a decapsulation oracle, which, when queried on an encapsulation C, returns its corresponding decapsulated symmetric key K.
Challenge: the challenger $\mathcal{C}$ computes $(K_0, C^*) \leftarrow \texttt{Encaps}(pk)$ and selects uniformly at random K_1 from the symmetric key space. Then $\mathcal{C}$ selects a bit $b \in \{0, 1\}$ uniformly at random and hands (K_b, C^*) to $\mathcal{A}$.
Queries: the adversary $\mathcal{A}$ is given access to a decapsulation oracle, which, when queried on an encapsulation C, returns its corresponding decapsulated symmetric key K. The challenge encapsulation C^* must not be queried by the decapsulation oracle.
Guess: the adversary outputs $b' \in \{0, 1\}$ and wins if and only if $b = b'$, and C^* was not queried by the decapsulation oracle.

Figure 1. IND-CCA security of a key encapsulation mechanism.

2.2. Signature Scheme

To formalize a signature scheme, we follow the usual approach.

Definition 3 (Signature). *A signature is a triple of polynomial time algorithms* (`SigKeyGen`, `Sign`, `Verify`) *as follows:*

- `SigKeyGen` *is probabilistic. Given the security parameter ℓ, it generates a pair of public and secret keys $(vk, sigk)$.*
- `Sign` *is probabilistic. Given a secret key sigk and a message M it generates a signature σ.*
- `Verify` *is deterministic. Given a public key vk, a signature σ, and a message M, this algorithm outputs* 1 *if the signature is valid and* 0 *otherwise.*

We require that for all key pairs $(vk, sigk)$ generated by `SigKeyGen` *and for all messages M the following correctness condition holds: If σ is an output of* $\texttt{Sign}(sigk, M)$*, then* $\texttt{Verify}_{vk}(\sigma, M) = 1$.

The security notion usually required for signatures is existential unforgeability under chosen message attacks, and we capture it by the following definition.

Definition 4 (EUF-CMA security). *A signature scheme is EUF-CMA secure if the advantage of any probabilistic polynomial time adversary $\mathcal{A}$ in the game described in Figure 2 is negligible. Here the advantage of an adversary $\mathcal{A}$ is defined as the function* $\mathrm{Adv}_{\mathcal{A}}^{\text{EUF-CMA}}(\ell) = \Pr[\mathrm{Succ}_{\mathcal{A}}^{\text{EUF-CMA}}]$, *where* $\mathrm{Succ}_{\mathcal{A}}^{\text{EUF-CMA}}$ *is the event that $\mathcal{A}$ wins.*

- Setup: the challenger $\mathcal{C}$ runs the key generation $\texttt{SigKeyGen}$ to obtain a key pair $(vk, sigk)$ and hands the public key vk to $\mathcal{A}$.
- Queries: the adversary $\mathcal{A}$ is given access to a signing oracle. When queried for message m, the challenger runs algorithm $\texttt{Sign}$ on input m and $sigk$ and returns the corresponding output σ to the adversary.
- Forgery: the adversary $\mathcal{A}$ outputs a pair (μ, σ) and wins if and only $\texttt{Verify}_{vk}(\sigma, \mu) = 1$ and no query for a signature on μ was asked.

Figure 2. EUF-CMA security of a signature scheme.

3. Security Model and Security Goals

The security model we use follows the oracle-based approach to capture adversarial capabilities—this approach is also used by Katz and Yung [1], for instance, building on the work of Bresson et al., work [13].

3.1. Protocol Participants

Given a security parameter ℓ, we assume that the set of protocol participants $\mathcal{U}$ is polynomial in ℓ. We model each user $U \in \mathcal{U}$ as a probabilistic polynomial time algorithm which is allowed to execute a polynomial number of protocol instances Π_U^s $(s \in \mathbb{N})$ concurrently. We assume user identities to be bitstrings of identical length ℓ and to keep notation simple, throughout we will use U referring to both the bitstring identifying a user U and the algorithm U itself. Let Π_U^s be a protocol instance, the following seven variables are associated with it:

pid_U^s: stores the identities of those users in $\mathcal{U}$with which a key is to be established a particular instance aims at establishing a key with, including U (this ensures being partnered is thus a reflexive relation);

sid_U^s: stores a session identifier, i.e., a non-secret public identifier for the session key sk_U^s;

sk_U^s: stores a distinguished NULL value and after a successful protocol run holds the session key;

acc_U^s: is set to TRUE if the session key stored in sk_U^s has been accepted;

state_U^s: keeps state information needed while executing the protocol (e.g., the secret scalars used as ephemeral Diffie–Hellman keys);

term_U^s: is set to TRUE if this protocol execution has terminated;

used_U^s: indicates if this instance is used, i.e., currently involved in a protocol execution.

3.2. Initialization

Before actual protocol executions take place, we allow an optional trusted initialization phase without adversarial interference. During this phase, for each user public and private key pairs (pk_U, ak_U) can be generated and distributed accordingly. These private and public keys can include the ones corresponding to a KEM, a signature scheme, etc. The initialization phase can also be used to issue (possibly needed) further public parameters.

3.3. Adversarial Capabilities and Communication Network

The adversary $\mathcal{A}$ is represented as a probabilistic polynomial time algorithm with full control over the communication network. The network is, therefore, fully asynchronous, non-private, allowing arbitrary point-to-point connections among users. More specifically, we express the capabilities of the adversary through the following oracles:

$\mathsf{Send}(U, s, M)$: This oracle can be used in two ways.

- The adversary can initialize a protocol execution; sending the special message $M = \{U_{i_1}, \ldots, U_{i_r}\} \subseteq \mathcal{U}$ to an unused instance Π_U^s with $U \in M$ initializes a protocol run among $U_{i_1}, \ldots, U_{i_r}$. After such a query, Π_U^s sets $\mathsf{pid}_U^s := \{U_{i_1}, \ldots, U_{i_r}\}$, $\mathsf{used}_U^s := \text{TRUE}$, and processes the first step of the protocol.

- The message M is sent to instance Π_U^s. The oracle returns the protocol message output by Π_U^s after receiving M.

$\mathsf{Reveal}(U, s)$: returns the stored session key sk_U^s if $\mathsf{acc}_U^s = \text{TRUE}$ and a NULL value otherwise.
$\mathsf{Corrupt}(U)$: for a user $U \in \mathcal{U}$ this query returns U's long-term secret key ak_U.

Notice that, unlike Reveal, Corrupt refers to a user instead of to an individual protocol instance.

We consider an adversary to be active if it has access to all of the above oracles. To capture a passive adversary, we can replace access to Send oracle with access to an Execute oracle, which returns a complete protocol transcript among the specified unused instances. An active adversary can simulate this Execute oracle using Send in the natural way.

For technical reasons, we introduce one more oracle, Test, and $\mathcal{A}$ must submit exactly one query of the form $\mathsf{Test}(U, s)$ with an instance Π_U^s that has accepted a session key, i.e., with $\mathsf{acc}_U^s = \text{TRUE}$. In response to such a query, a bit $b \leftarrow \{0, 1\}$ is sampled uniformly at random. For the case $b = 1$, the established session key stored in sk_U^s is returned. For $b = 0$ the output is a uniformly random element sampled from the space of session keys. The idea is that for a secure group key establishment protocol, no efficient adversary can distinguish between the cases $b = 0$ and $b = 1$. To turn this idea into a definition, we exclude trivial cases and focus on correct group key establishment protocols:

Definition 5 (Correctness). *A group key establishment is said to be correct if on honest delivery of all messages and all users being honest, a single protocol execution among users $U_0, \ldots, U_{n-1}$ involves n instances $\Pi_0^{s_0}, \ldots, \Pi_{n-1}^{s_{n-1}}$ such that with overwhelming probability all of the following hold:*

- *all users accept, i. e.,* $\mathsf{acc}_0^{s_0} = \cdots = \mathsf{acc}_{n-1}^{s_{n-1}} = \text{TRUE}$;
- *all users obtain the same session identifier, i. e.,* $\mathsf{sid}_0^{s_0} = \cdots = \mathsf{sid}_{n-1}^{s_{n-1}}$;
- *all users accept the same session key, i. e.,* $\mathsf{sk}_0^{s_0} = \cdots = \mathsf{sk}_{n-1}^{s_{n-1}} \neq \text{NULL}$ *associated with the same session identifier* $\mathsf{sid}_0^{s_0}$;
- *all communication partners are specified as desired communication partner, i. e.,* $\mathsf{pid}_0^{s_0} = \cdots = \mathsf{pid}_{n-1}^{s_{n-1}} = \{U_0, \ldots, U_{n-1}\}$.

Correctness refers to a scenario where no attack takes place; to formulate security guarantees we need to specify the circumstances under which a correct guess for the random bit used by the Test oracle constitutes a possible attack. For this we use the following notions of partnering and freshness.

Definition 6 (Partnering). *Two instances $\Pi_{U_i}^{s_i}$ and $\Pi_{U_j}^{s_j}$ are partnered if* $\mathsf{sid}_{U_i}^{s_i} = \mathsf{sid}_{U_j}^{s_j}$, $\mathsf{pid}_{U_i}^{s_i} = \mathsf{pid}_{U_j}^{s_j}$, *and* $\mathsf{acc}_{U_i}^{s_i} = \mathsf{acc}_{U_j}^{s_j} = \text{TRUE}$.

Based on this notion, we can determine what a fresh instance is, i.e., an instance where the adversary does not know the session key for trivial reasons.

The following formulation allows an adversary $\mathcal{A}$ to reveal all secret keys without violating freshness, provided $\mathcal{A}$ does not send any "relevant" messages afterwards. Therefore, security in the sense of Definition 8 below implies forward secrecy:

Definition 7 (Freshness). *An instance $\Pi_i^{s_i}$ is called fresh if none of the following two conditions hold:*

- *For some $U_j \in \mathsf{pid}_i^{s_i}$ a* $\mathsf{Corrupt}(U_j)$ *query was executed before a query of the form* $\mathsf{Send}(U_k, s_k, *)$ *has taken place where $U_k \in \mathsf{pid}_i^{s_i}$.*
- *A query* $\mathsf{Reveal}(U_j, s_j)$ *with $\Pi_i^{s_i}$ and $\Pi_j^{s_j}$ being partnered occurred.*

We write $\mathsf{Succ}_{\mathcal{A}}^{\mathsf{ke}}$ for the event that $\mathcal{A}$ queries Test with a fresh instance according to Definition 7 and outputs a correct guess for the Test oracle's bit b.

Definition 8 (Semantic security)**.** *A key establishment protocol is said to be semantically secure, if the advantage* $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{ke}} = |2 \cdot \Pr[\mathsf{Succ}_{\mathcal{A}}^{\mathsf{ke}}] - 1|$ *is negligible for all probabilistic polynomial time algorithms* $\mathcal{A}$.

To make explicit that adversaries are considered to be active, i.e., have access to the Send oracle, it is common to refer to a group key establishment protocol as authenticated. On the other hand, we speak of unauthenticated group key establishment if the adversaries are passive.

4. Proposed Construction

The proposed generic construction for designing an authenticated group key establishment for n users $U_0, \dots, U_{n-1}$ is shown in Figure 3. It builds on an available key encapsulation mechanism and a signature scheme. The following theorem shows that the proposed construction offers a provable security guarantee, if the underlying primitives meet standard security requirements.

Set up: A pair of keys $(vk_i, sigk_i)$ for the signature scheme $\mathcal{S}$ is generated for each U_i, which gets the secret key $sigk_i$ while vk_i is publicized.

Round 1: Computation. Each U_i generates an ephemeral pair of keys for $\mathcal{E}$:

$$(pk_i, sk_i) \leftarrow \texttt{KeyGen}(1^\ell)$$

Broadcast. Each U_i sends to his right[a] neighbor the message (pk_i, U_{i+1})

Round 2: Computation. Each U_i, using the ephemeral public key he has received, computes an encapsulation for that key:

$$(C_{i-1}, K_{i-1}) \leftarrow \texttt{KeyEncaps}(pk_{i-1});$$

Broadcast. Each U_i does sends to his left neighbor the message (C_{i-1}, U_{i-1})

Round 3: Computation. Each U_i does the following:

- Using his ephemeral secret key, decapsulates the received key $K_i \leftarrow \texttt{KeyDecaps}_{sk_i}(C_i)$;
- Using the encapsulated key he sent before, computes $X_i = K_{i-1} \oplus K_i$;
- Computes a signature σ_i on $pk_0, \dots, pk_{n-1}, C_0, \dots, C_{n-1}, X_i, pid_i$

Broadcast. Each U_i broadcasts (X_i, σ_i)

Key Computation. Check. Each U_i checks all the signatures, equality of pids, $X_0 \oplus \cdots \oplus X_{n-1} = 0$; if any check fails, aborts.

Computation. Each U_i does the following:

- For $j = 0, \dots, n-1$, computes $K_j = K_i \oplus \bigoplus_{h=j+1}^{i} X_h$;
- Sets the session identifier sid_i to $pk_0, \dots, pk_{n-1}, C_0, \dots, C_{n-1}, X_0, \dots, X_{n-1}, pid_i$.;
- Accepts K_0 as session key.

[a] The ordering can be determined by means of lexicographic ordering of user identifiers, for instance, and the users are assumed to be in a circle, i.e., U_0 and U_{n-1} are neighbours

Figure 3. A compiler achieving authenticated group key establishment from a secure key encapsulation mechanism (KEM) and a secure signature scheme.

Theorem 1. *Assuming $\mathcal{S}$ is an EUF-CMA secure signature scheme and $\mathcal{E}$ is an IND-CCA secure KEM, the protocol from Figure 3 is correct and semantically secure.*

Proof. We can easily verify the protocol correctness: if all the participants follow the protocol description and there is no active adversarial interference, then all checks will succeed and every participant will set the same pid and sid. Moreover every participant will receive the correct $\{X_j\}_{j=0}^{n-1}$ and consequently they will be able to compute the same session key K_0.

To illustrate the security of our compiler, we use the "game hopping" technique, where we let the adversary $\mathcal{A}$ interact with a simulator $\mathcal{B}$. We denote by $\mathrm{Adv}(\mathcal{A}, G_i)$ the advantage of the adversary in Game i. The security parameter is denoted by ℓ. Further, we denote by q_e and q_s the maximum number of queries made by the adversary to the Execute and Send oracles, respectively.

Game 0. This game is identical to the original attack game, with all the oracles being simulated as in the real protocol. Therefore,

$$\mathrm{Adv}(\mathcal{A}, G_0) = \mathrm{Adv}_{\mathcal{A}}^{\mathsf{ke}}(\ell).$$

Game 1. Let Forge be the event that the adversary succeeds in forging an authenticated message $pk_0, \ldots, pk_{n-1}, C_0, \ldots, C_{n-1}, X_i, pid_i, \sigma_i$ of at least one party U_i without having queried $\mathsf{Corrupt}(U_i)$ and where all the values signed involved $pk_0, \ldots, pk_{n-1}, C_0, \ldots, C_{n-1}, X_i, pid_i, \sigma_i$ were not output by a same U_i's instance. Any time this event occurs, we abort and mark this as success for the adversary.

An adversary $\mathcal{A}$ that can achieve Forge can be used to construct an adversary $\mathcal{A}'$ that forges a signature in the EUF-CMA game: the given public key is assigned randomly to U_i, one of the users; all other participants are initialized as the protocol indicates; afterwards all the queries in the security game are answered faithfully and when a signature by the chosen user is needed, the signing oracle of the EUF-CMA game is queried to produce it.

The probability of the adversary choosing U_i when assigning the public key for the signature is at least $1/|\mathcal{U}|$, and with $|\mathcal{U}|$ being polynomial size, this is non-negligible:

$$\mathrm{Adv}_{\mathcal{A}'}^{\text{EUF-CMA}} \geq \frac{1}{|\mathcal{U}|} \cdot \Pr(\mathsf{Forge}),$$

which yields

$$|\mathrm{Adv}(\mathcal{A}, G_0) - \mathrm{Adv}(\mathcal{A}, G_1)| \leq \mathrm{poly}_{\mathsf{Forge}}(\ell) \cdot \mathrm{Adv}_{\mathcal{A}'}^{\text{EUF-CMA}},$$

for a polynomial bound $\mathrm{poly}_{\mathsf{Forge}}$ on $|\mathcal{U}|$.

Game 2. This game is exactly as Game 1 except that a session t is chosen uniformly at random. If the Test query does not occur in the t-th session the game aborts, and we count it as win for the adversary. As the number of active protocol instances is polynomially bounded, we have

$$\mathrm{Adv}(\mathcal{A}, G_1) \geq \mathrm{poly}_{\mathsf{Test}}(\ell) \cdot \mathrm{Adv}(\mathcal{A}, G_2),$$

for a polynomial bound $\mathrm{poly}_{\mathsf{Test}}$ on the number of protocol sessions activated.

Game 3. This game is identical to the previous game, except that the simulation of the Send, Execute and Test oracles is modified as follows. The symmetric key K_0 output by $\mathtt{Encaps}(pk_0)$ in the Test instance Π_0^t is replaced with a random key K^* chosen from $\{0,1\}^{\ell}$.

In order to bound the difference in the advantages between Games 2 and 3, we will build, from an adversary $\mathcal{A}$ ble to distinguish between both games, an adversary $\mathcal{B}$ attacking the key encapsulation mechanism $\mathcal{E}$ such that

$$|\mathrm{Adv}(\mathcal{A}, G_2) - \mathrm{Adv}(\mathcal{A}, G_3)| \leq 2 \cdot \mathrm{Adv}_{\mathcal{B}}^{\text{IND-CCA}}(\ell),$$

where $\mathrm{Adv}_{\mathcal{B}}^{\text{IND-CCA}}(\ell)$ denotes the advantage of a probabilistic polynomial time adversary $\mathcal{B}$ attacking $\mathcal{KEM}$.To establish this bound, we assume that $\mathcal{B}$, which runs $\mathcal{A}$ as an auxiliary algorithm, can access a simulation of $\mathcal{KEM}$. Further, $\mathcal{B}$ executes the key generation algorithm of $\mathcal{S}$ for each user U_i, thus obtaining a pair of keys $(vk_i, sigk_i)$ for the signature scheme. Adversary $\mathcal{B}$ also executes the key generation algorithm of $\mathcal{KEM}$ for user U_0, and obtains the public key corresponding to users $U_1, \ldots, U_n$. Our adversary $\mathcal{B}$ obtains a challenge (C^*, K_0, K_1) as described in Definition 1, and we have to describe how $\mathcal{B}$ answers to $\mathcal{A}$'s queries:

- Whenever a query $\mathsf{Corrupt}(U_i)$ is made by $\mathcal{A}$, $\mathcal{B}$ generates the keys for the signature and returns $sigk_i$ as answer to $\mathcal{A}$.
- To answer a Send query for Round 2 involving U_0, $\mathcal{B}$ uses the challenge encapsulation C^*. The rest of the answers are generated as in a real execution of the protocol.
- To answer an Execute query by $\mathcal{A}$, our adversary$\mathcal{B}$ modifies the messages as described for the simulation of the Send oracle.
- A Reveal query by $\mathcal{A}$ is answered in a similar way as a Send or Execute query. Notice that a Reveal query cannot be made on t or any partnered instance. To answer any other Reveal query $\mathcal{B}$ uses the decapsulation oracle of its IND-CCA game.
- Finally, to answer a Test query, a bit b' is chosen by $\mathcal{B}$ when starting the simulation. $\mathcal{B}$ will return the key $K_{b'}$ received from the KEM challenger to $\mathcal{A}$.

At some point $\mathcal{A}$ will output a bit b'' as a guess for b' which will determine the output b of $\mathcal{B}$ for the KEM challenge. Specifically, $\mathcal{B}$ outputs $b = 0$ if and only if $b' = b''$. Taking into account that the view of $\mathcal{A}$ is identical to Game 2, if the answers of $\mathcal{B}$'s simulation of Test are real keys and to Game 3 if the answers of $\mathcal{B}$'s simulation of Test are random ones, we obtain that $|\mathrm{Adv}(\mathcal{A}, G_2) - \mathrm{Adv}(\mathcal{A}, G_3)|$ is bounded by $2 \cdot \mathrm{Adv}_{\mathcal{B}}^{IND-CCA}$.

To conclude the proof, we can see that the advantage of the adversary in Game 3 equals 0, as the session keys are chosen uniformly at random in $\{0,1\}^{\kappa_\ell}$. Collecting all the advantages, we can see that $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{ke}}$ is indeed negligible. □

4.1. Remark

We would like to point out that our scheme, as described above, meets precisely the security criteria described in Section 3. It would be possible to adjust our construction to capture additional security notions, for instance introducing contributory properties. In this case, only a minor modification becomes necessary. In fact, since all of $K_0, \ldots, K_{n-1}$ are available to each legitimate user, one could choose $K_0 \oplus \cdots \oplus K_{n-1}$ as session key, rather than just K_0. The proof would then be amended accordingly, as follows: in Game 3, after replacing K_0 with K^*, produce a pseudorandom session key as $K^* \oplus \cdots \oplus K_{n-1}$. The pseudorandomness follows immediately from the fact that K^* is chosen uniformly at random in $\{0,1\}^{\ell}$.

4.2. Instantiation

To instantiate the above AGKE construction, there are various natural options both for KEMs and signature schemes. Some possible candidates for quantum-safe KEMs include [14–17]. Moreover, examples for candidates of post-quantum signature schemes are offered by [9,18]. The particular choice of schemes can be made taking into account their efficiency or the assumption their security relies on. Different applications may have different preferences for prioritizing, e. g., public-key size over signature size.

5. Conclusions

The protocol compiler presented here offers a convenient approach to systematically design authenticated group-key establishment protocols in a post-quantum scenario. Keeping the number of signatures needed low (one per user), gives a protocol designer flexibility in the type of post-quantum signature signature scheme to be deployed. With three rounds, the round complexity that is achievable with currently available key encapsulation mechanisms appears quite attractive, too.

Author Contributions: Conceptualization, E.P., R.S. and A.S.C.; methodology, E.P., R.S. and A.S.C.; validation, E.P., R.S. and A.S.C.; formal analysis, A.S.C.; investigation, E.P., R.S. and A.S.C.; resources, R.S. and A.S.C.; writing—original draft preparation, E.P., R.S. and A.S.C.; writing—review and editing, E.P., R.S. and A.S.C.; project administration, R.S. and A.S.C.; funding acquisition, R.S. and A.S.C.

Funding: This research was funded in part by the NATO Science for Peace and Security Programme under grant G5448 and through research project MTM2017-83506-C2-2-P by the Spanish MICINN.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Katz, J.; Yung, M. Scalable Protocols for Authenticated Group Key Exchange. *J. Cryptol.* **2007**, *20*, 85–113. [CrossRef]
2. Apon, D.; Dachman-Soled, D.; Gong, H.; Katz, J. Constant-Round Group Key Exchange from the Ring-LWE Assumption. In Proceedings of the 10th International Conference on Post-Quantum Cryptography (PQCrypto 2019), Chongqing, China, 8–10 May 2019; pp. 189–205.
3. Bresson, E.; Manulis, M.; Schwenk, J. On Security Models and Compilers for Group Key Exchange Protocols. In Proceedings of the Second International Workshop on Security (IWSEC), Nara, Japan, 29–31 October 2007; pp. 292–307. [CrossRef]
4. Bohli, J. A Framework for Robust Group Key Agreement. In Proceedings of the International Conference on Computational Science and Its Applications—ICCSA 2006, Glasgow, UK, 8–11 May 2006; pp. 355–364. [CrossRef]
5. Burmester, M.; Desmedt, Y. A Secure and Efficient Conference Key Distribution System (Extended Abstract). In Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques (UROCRYPT '94), Perugia, Italy, 9–12 May 1994; pp. 275–286. [CrossRef]
6. Shor, P.W. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; pp. 124–134. [CrossRef]
7. Bundesamt für Sicherheit in der Informationstechnik. Studie: Entwicklungsstand Quantencomputer V.1.1. 2019. Available online: https://www.bsi.bund.de/DE/Publikationen/Studien/Quantencomputer/quantencomputer.html (accessed on 28 November 2019).
8. National Academies of Sciences, Engineering, and Medicine. *Quantum Computing: Progress and Prospects*; The National Academies Press: Washington, DC, USA, 2019. [CrossRef]
9. Bernstein, D.J.; Dobraunig, C.; Eichlseder, M.; Fluhrer, S.; Gazdag, S.L.; Hülsing, A.; Kampanakis, P.; Kölbl, S.; Lange, T.; Lauridsen, M.M.; et al. SPHINCS$^+$. Submission to the NIST Post-Quantum Project. 2017. Available online: https://sphincs.org/data/sphincs+-submission-nist.zip (accessed on 28 November 2019).
10. Tang, Q.; Mitchell, C.J. Efficient Compilers for Authenticated Group Key Exchange. In Proceedings of the International Conference on Computational Intelligence and Security (CIS), Xi'an, China, 15–19 December 2005; pp. 192–197. [CrossRef]
11. Post-Quantum Cryptography Standardization. 2019. Available online: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization (accessed on 28 November 2009).
12. Shoup, V. Using hash functions as a hedge against chosen ciphertext attack. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Bruges, Belgium, 14–18 May 2000; pp. 275–288.
13. Bresson, E.; Chevassut, O.; Pointcheval, D.; Quisquater, J. Provably authenticated group Diffie-Hellman key exchange. In Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001), Philadelphia, PA, USA, 6–8 November 2001; pp. 255–264. [CrossRef]
14. Barreto, P.S.L.M.; Gueron, S.; Güneysu, T.; Misoczki, R.; Persichetti, E.; Sendrier, N.; Tillich, J. CAKE: Code-Based Algorithm for Key Encapsulation. In Proceedings of the 16th IMA International Conference on Cryptography and Coding (IMACC), Oxford, UK, 12–14 December 2017; pp. 207–226. [CrossRef]
15. Banegas, G.; Barreto, P.; Boidje, B.O.; Cayrel, P.; Dione, G.N.; Gaj, K.; Gueye, C.T.; Haeussler, R.; Klamti, J.B.; Ndiaye, O.; et al. DAGS: Key encapsulation using dyadic GS codes. *J. Math. Cryptol.* **2018**, *12*, 221–239. [CrossRef]

16. Bos, J.W.; Costello, C.; Ducas, L.; Mironov, I.; Naehrig, M.; Nikolaenko, V.; Raghunathan, A.; Stebila, D. Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–29 October 2016; pp. 1006–1018. [CrossRef]
17. Bos, J.W.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS—Kyber: A CCA-Secure Module-Lattice-Based KEM. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P 2018), London, UK, 24–26 April 2018; pp. 353–367. [CrossRef]
18. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 238–268. [CrossRef]