Many a Mickle Makes a Muckle: A Framework for Provably Quantum-Secure Hybrid Key Exchange

Many a Mickle Makes a Muckle: A Framework for Provably Quantum-Secure Hybrid Key Exchange

Benjamin Dowling[1], Torben Brandt Hansen[2], and Kenneth G. Paterson[1]

[1] Department of Computer Science, ETH Zurich.
[2] Information Security Group, Royal Holloway, University of London.
benjamin.dowling@inf.ethz.ch, Torben.Hansen.2015@rhul.ac.uk,
kenny.paterson@inf.ethz.ch

Abstract. Hybrid Authenticated Key Exchange (AKE) protocols combine keying material from different sources (post-quantum, classical, and quantum key distribution (QKD)) to build protocols that are resilient to catastrophic failures of the different components. These failures may be due to advances in quantum computing, implementation vulnerabilities, or our evolving understanding of the quantum (and even classical) security of supposedly quantum-secure primitives. This hybrid approach is a prime candidate for initial deployment of post-quantum-secure cryptographic primitives because it hedges against undiscovered weaknesses. We propose a general framework HAKE for analysing the security of such hybrid AKE protocols. HAKE extends the classical Bellare-Rogaway model for AKE security to encompass forward security, post-compromise security, fine-grained compromise of different cryptographic components, and more. We use the framework to provide a security analysis of a new hybrid AKE protocol named Muckle. This protocol operates in one round trip and leverages the pre-established symmetric keys that are inherent to current QKD designs to provide message authentication, avoiding the need to use expensive post-quantum signature schemes. We provide an implementation of our Muckle protocol, instantiating our generic construction with classical and post-quantum Diffie-Hellman-based algorithmic choices. Finally, we report on benchmarking exercises against our implementation, examining its performance in terms of clock cycles, elapsed wall-time, and additional latency in both LAN and WAN settings.

Keywords: Authenticated key exchange, hybrid key exchange, provable security, protocol analysis, quantum key distribution, post-compromise security

1 Introduction

NIST's Post Quantum Cryptography (PQC) process has triggered significant effort into the design of new post-quantum public key algorithms that can eventually be used to replace existing algorithms in protocols such as IPsec and TLS. Indeed, NIST's 2017 call received 69 complete submissions in various categories.

However, much less attention has been paid on how to securely integrate these new algorithms into applications, and to assessing the impact they will have on the performance of real-world network protocols. A key issue is that the new algorithms are relatively immature, and our understanding of their security is still evolving. NIST lacked confidence in 13 of the original submissions [22]; meanwhile Albrecht et al. [4] highlight how poor our current understanding is of how to assess the cost of lattice attacks. During the cryptographic interregnum, sensitive data is still at risk from attackers who are willing to record and store network traffic for later cryptanalysis. One response to this uncertainty is to quickly roll out post-quantum secure algorithms in protocols like TLS. For example, in 2016 Google carried out an experiment in which they deployed the NewHope lattice-based scheme [5] in Chrome and in Google servers [13], and in 2019 Cloudflare and Google jointly carried out similar experiments [20]. These tests adopted hybrid approaches, combining post-quantum schemes with forward-secure key exchange mechanisms, namely Elliptic Curve Diffie Hellman Ephemeral (ECDHE). Adopting a hybrid approach hedges against security vulnerabilities in the post-quantum algorithm (fundamental as well as implementation-related) whilst providing security against quantum adversaries. While discussions have started [27], at this point no formal standardisation has begun integrating post-quantum algorithms into secure Internet protocols, a few unadopted IETF drafts notwithstanding [25, 29]. Standardisation will inevitably be needed, and we anticipate that a hybrid approach will be used. But first the community needs to research a) how to build and analyse hybrid protocols, and b) how to assure the security of their post-quantum components. The former is the main focus of this work, while the latter falls under the aegis of the NIST PQC process.

Quantum Key Distribution (QKD) is often promoted as an alternate solution to the threat posed by large-scale quantum computers, and has some attractive features: when well-implemented, it can offer unconditional security, it is also increasingly well-integrated with standard optical communications and electronics systems, with small package sizes and high raw bit rates, cf. [26]. However, the achievable bit rate does not yet practically allow the use of QKD keying material in a one-time-pad encryption system, so while the keying material may be unconditionally secure, no practical overall secure communications system relying on QKD is (to date). Moreover, QKD is fundamentally range limited (in the absence of quantum repeaters) and so cannot offer true end-to-end security in wide-area networks. Furthermore the technology is still quite immature, and vulnerable to various implementation attacks ("quantum hacking"), cf. [28, 18]. Even the physical basis of QKD has been questioned [30, 10]. Despite this, QKD may still usefully augment existing technologies in point-to-point applications, such as intra or inter data-centre communications or in metropolitan networks. Given this context, we should consider the possibility of incorporating QKD-based keying material into our hybrid protocol designs, resulting in three sources of keying material to combine: classically-secure (e.g. ECDHE), post-quantum secure (e.g. NewHope, SIDH, or another NIST candidate), and QKD-based. Having established this context, we can now begin to describe our contributions.

1.1 Our Contributions

The HAKE security framework: We introduce a flexible framework for capturing and analysing Hybrid Authenticated Key Exchange (AKE) protocols that combine a wide variety of symmetric and asymmetric primitives. The HAKE framework is the result of heavily modifying the classic Bellare-Rogaway [7] model for AKE, incorporating security notions such as perfect forward secrecy and post-compromise security (referring to the ability of a key exchange protocol to recover security in the event of a catastrophic compromise of all its secrets) and smoothly caters for different strengths of adversary (quantum or even classical). It features a particularly simple and novel abstraction of QKD protocols to allow them to be modelled in a standard computational setting: pairs of parties are given private access to a shared source of secret random bits.

The Muckle AKE protocol: To exercise the HAKE framework, we also present the Muckle AKE protocol,[3] its security analysis, details of a working software implementation of Muckle, and benchmarking results. Muckle securely combines keying material obtained from a quantum key distribution (QKD) protocol with that from a post-quantum-secure key encapsulation mechanism (KEM) and a classically-secure KEM. Muckle is a one-round (1-RTT) protocol which exploits the presence of a QKD component to simplify the authentication of protocol messages. Specifically, QKD protocols typically assume the presence of an initial or pre-shared key (PSK) between the pair of communicating parties. This is used to bootstrap an authenticated channel for exchanging basis measurement information.[4] Muckle's design assumes the presence of a second PSK (since the cost of establishing two such keys is not any greater in practice than the cost of establishing just one), and uses it as the basis for authenticating its protocol messages via MACs. Muckle evolves this key and associated state, using outputs from the various KEM primitives as well as the QKD itself.

Benchmarking Muckle: We instantiate and implement Muckle in 'C' (which we denote C-Muckle) and benchmarked it in different network settings, selecting specific schemes in order to fix a concrete design. We profile the cost of the underlying C-Muckle functions in terms of the median execution wall-time and clock cycle counts. We also contrast the wall-time profiling of C-Muckle functions when it runs over a LAN with the same profiling when it is run between London and Paris (approximately 500km, somewhat more than the current maximum range of single-hop QKD systems). These experiments are done without a real QKD system, which is simulated via access to a file of keying material.

Security analysis of Muckle: Finally, we demonstrate that Muckle achieves AKE security as defined by our HAKE framework. This allows us to make security statements about Muckle in the presence of quantum adversaries (assuming post-quantum variants of standard cryptographic assumptions), or under the catastrophic failure of all but one of its distinct components. The latter includes

[3] The name Muckle derives from the traditional English phrase "Many a mickle makes a muckle": many small things can add up to make a big thing.

[4] As a side-note, this is why QKD in this normal form does not solve the key distribution problem, but only the key expansion problem.

scenarios where, for example, all public key cryptography evaporates (and only Muckle's QKD component remains secure). It also includes the situation where the QKD component turns out to be badly engineered and therefore insecure and where the classical component becomes vulnerable to a quantum computer, but where its post-quantum counterpart remains secure.

1.2 Related Work

While the analysis of "fully classical" hybrid schemes have appeared in the past (for instance, work on combining multiple public-key encryption schemes [31]), little work has been done on combining post-quantum and classical cryptographic primitives. Bindel et al. [12] examine a variety of hybrid digital signature schemes in quantum and post-quantum settings. They also formalise the notion of *separability*, which captures the ability of an attacker to separate the hybrid scheme into its individual cryptographic components. Bindel et al. [11] is most closely related to our work, considering hybrid key exchange in a similar setting to our own, but is focussed on quantum-secure KEM combiners. Their setting and security model are less general than ours in some regards (our HAKE framework can accommodate KEMs, theirs is limited to KEMs), but considers a heirarchy of attackers depending on quantum-computing capability and quantum access to the protocol participants. In addition, their compromise paradigm is less fine-grained, considering only the compromise of long-term and session keys. Complementing our approach, Mosca et al. [23] analyse the security of the QKD protocol BB84 [8], using an AKE security model in the tradition of Bellare-Rogaway to formalise the protocol in their notation. They prove the security of BB84 in this security model, and their notions of keys output by the QKD protocol match our assumptions. The concept of *breakdown resilience* was introduced by Brendel et al. [14]; this concept considers the effect on overall protocol security of failures of individual cryptographic components. They also extend Bellare-Rogaway security models by providing an interface for an attacker to break individual cryptographic components, similar to our approach of providing specific key exposure oracles. There have been a couple of recent IETF drafts [25, 29] describing hybrid approaches for TLS 1.3, but without any accompanying formal security analysis as far as we are aware.

2 The Muckle Protocol

Here we introduce the Muckle hybrid key exchange protocol; see Figure 1 for an overview. At a high-level, Muckle simultaneously executes post-quantum and classical key encapsulation primitives, and draws key material from a QKD protocol, represented abstractly in the protocol as a shared array of bits into which the two parties can index. The three distinct types of key material are used as inputs to a sequence of key derivation steps that we refer to as the Muckle key schedule. The design of the Muckle key schedule allows us to prove that the

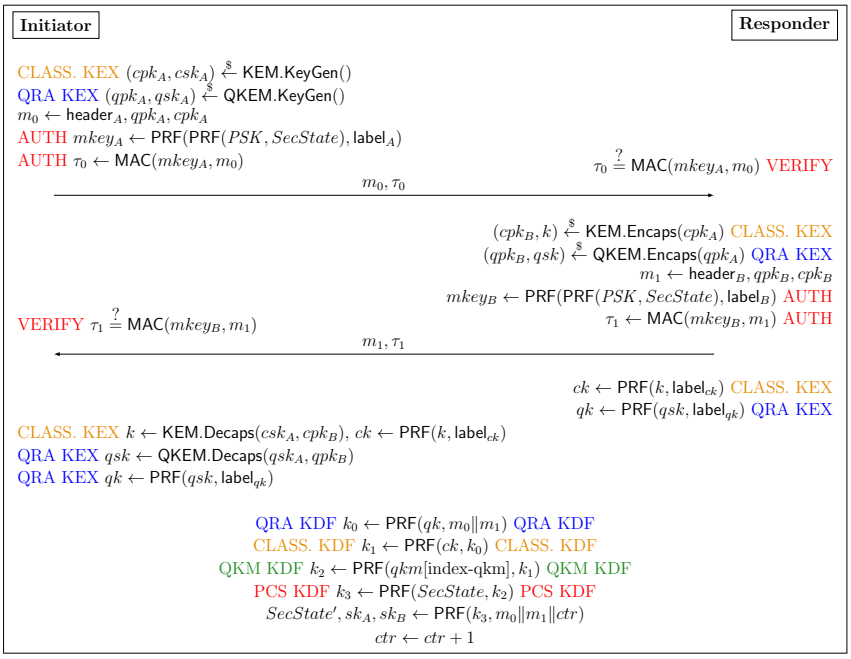

Fig. 1: A single stage of the Muckle protocol.

session keys produced by Muckle are resilient to vulnerabilities in the underlying QKD or key exchange primitives. Muckle is a multi-stage protocol, where the initiator and responder repeatedly run the single stage shown in Figure 1, updating the session keys sk_A, sk_B and the secret, shared state $SecState$ of the protocol at each stage. We highlight the key features of Muckle below:

- One round trip (1-RTT) to establish post-quantum-secure session keys.
- Multi-stage design and the inclusion of an updating secret state ($SecState$) allows Muckle to achieve post-compromise security, i.e. recover security after full compromise attacks(under certain restrictions, see Section 4.)
- Hybrid key exchange approach allows Muckle to be secure against classical adversaries even if the QKD and post-quantum components fail.
- Use of symmetric cryptography (of an appropriate key-length) allows Muckle to achieve post-quantum authentication without the use of computationally-expensive and bandwidth-intensive post-quantum signatures.
- Modular design allows implementers to easily replace underlying key exchange primitives if vulnerabilities are discovered.
- Key confirmation and full message transcript agreement of previous stages are provided in successive stages via the computation of authentication keys.

We expand on these below, and explain the different components of Muckle.

Message Structure: There are four elements to a Muckle message: a header (referred to in Figure 1 as header_A and header_B), containing message identifiers,

cryptographic primitive identifiers and party identifiers; a classical ephemeral key encapsulation, (which we instantiate with elliptic-curve-based Diffie-Hellman (ECDH) in C-Muckle); a post-quantum ephemeral key encapsulation, (which we instantiate with Supersingular Isogeny-based Diffie-Hellman (SIDH) in C-Muckle); and a MAC tag computed over the message.

QKD: Muckle assumes that a QKD scheme is running between pairs of communicating parties. QKD schemes make use of classically-authenticated communication channels, and such channels are (in practice) built using symmetric keys (though they could use other cryptographic techniques, such as digital signatures). Thus, Muckle assumes the presence of pre-shared symmetric keys (PSKs) between pairs of communicating parties. Likewise, this makes it possible to assume the existence of pre-established party identifiers in the protocol. Theses two value allow us to achieve post-quantum-secure authentication of the Muckle messages without incurring the significant computational or communication overhead that would be associated with a post-quantum signature scheme.

In our description of Muckle, we abstract the QKD protocol by modelling its output as an array of independent, uniformly-random bits (denoted $qsk[\cdot]$ in Figure 1) that is available to both parties in the protocol, otherwise treating the QKD component as a black box. Thus, we assume that the QKD system is implemented perfectly. This significantly simplifies our security analysis task, since it avoids the need for us to integrate existing QKD security models with our HAKE security framework. However, this is an idealisation that we plan to relax in future work, see Section 6 for more discussion. The pre-shared key is denoted PSK in Figure 1, and is 256 bits in size. The party identifiers are 32-byte strings (they do not appear explicitly in Figure 1, but instead are implicit in label_A and label_B).

Authentication: MAC tag computations use freshly-generated keys ($mkey_A$, $mkey_B$) for each new stage. Specifically, $mkey_A \leftarrow \mathsf{PRF}(\mathsf{PRF}(PSK,\ SecState), \mathsf{label}_A)$, and $mkey_B \leftarrow \mathsf{PRF}(\mathsf{PRF}(PSK, SecState), \mathsf{label}_B)$. Note that $SecState$ is updated with each new stage, and thus $mkey_A$ and $mkey_B$ are similarly fresh.

Key Schedule: The key schedule is run after the initiator and responder have sent and received their respective messages. The straightforward iterative design simplifies the analysis of the protocol. Each step takes as input some key material and a chaining key, and outputs a new chaining key used in the next iteration, seen in the KDF steps at the bottom of Figure 1. We also include a counter ctr (a 256-bit integer) in the final PRF computation; ctr is incremented after each stage.

State Update: The secret state $SecState$ is updated at the end of a Muckle stage (initialised as a constant, public value in the first stage), when the session keys are computed. Specifically, $SecState, sk_A, sk_B \leftarrow \mathsf{PRF}(k_3, m_0 \| m_1 \| ctr)$, taking as input the final chain key k_3 from the key schedule, and the concatenation of the message transcript and counter: $m_0 \| m_1 \| ctr$. Thus, consecutive Muckle stages provide implicit key confirmation (since in order to derive the same $SecState$, protocol participants must also derive the same session keys sk_A, sk_B) of previous stages, as well as full message transcript agreement of previous stages.

Post-Compromise Security (PCS): At a high-level, PCS is the ability of a key exchange protocol to recover security when an attacker has compromised all secrets of a session, if the attacker becomes passive in a later protocol execution. Our Muckle design achieves PCS by virtue of the inclusion of the secret state *SecState* in the MAC computations and in the derivation of the session keys.

3 Instantiation and Implementation of Muckle

This section describes our reference implementation of Muckle in 'C', which we denote C-Muckle [2]. C-Muckle follows the same governing design principles as Muckle, favouring simplicity and verifiability. As a result, we optimise for readability and reproducibility and sacrifice features such as fully performance-optimised code. C-Muckle targets 128-bit post-quantum security. To instantiate C-Muckle, we have made the following choice of parameters and cryptographic algorithms: For the classically-secure KEM, we use elliptic-curve Diffie-Hellman key exchange using the elliptic curve curve25519 [9], and for the post-quantum-secure KEM, we chose supersingular isogeny Diffie-Hellman key exchange using field arithmetic over the prime p503, construction and parameters by Costello et. al. [16]. The pseudo-random function is instantiated by the key derivation scheme HDKF [19] using 256-bit keys, and similarly the message authentication code is instantiated by HMAC [6] using 256-bit keys. For details about the C-Muckle message format, refer to the full version.

Dependencies: To provide support for the chosen cryptographic components C-Muckle relies upon two libraries: mbedtls [1] version 2.13.0 and PQCrypto-SIDH [3] version 3. The former is used to support the ECDHE, PRF, and MAC cryptographic components as well as random number generation, while the latter is used to support SIDH.

QKD Bits: Currently, our software implementation of C-Muckle does not engage with real QKD devices. The process of obtaining the bits produced by a QKD protocol is therefore emulated. We provide two distinct methods for doing this. The first method is to store a static array of bits in the source code. During an execution, bits are read from the array depending on an index. The second method reads from a file, with bits similarly read from the file depending on an index. In both cases the bits should be uniformly random. The method of emulation can be changed during compile-time. Currently, C-Muckle defaults to using the static array method. These methods are solely implemented for experimental use and should not be used in any production system. C-Muckle is designed to allow easy switching to a method that provides true access to QKD key material.

3.1 Performance Study

Here we profile and discuss the performance of C-Muckle. Our experiments aim at conveying the cryptographic costs associated with the different components

of Muckle, as well as the total cost of executing a complete run of the protocol. To achieve this, we benchmark different parts of C-Muckle as well as the core cryptographic API calls made to external libraries.

Methodology: We measure the performance of C-Muckle using two metrics: clock cycles and wall-time. For each metric, a single stage execution of C-Muckle is measured and recorded. The cost of lower layer functions responsible for performing cryptographic operations is also measured and separately recorded. Below we list these functions and describe the cryptographic operation they each perform:

muckle_ecdh_gen(): Generates an ECDHE public key pair.
muckle_ecdh_compute(): Computes the ECDHE secret.
muckle_sidh_gen(): Generates the SIDH public key pair.
muckle_sidh_compute(): Computes the SIDH shared secret.
muckle_read_qkd_keys(): Reads the QKD keying material using a method described above.
muckle_derive_keys(): Derives the secret state and session keys according to the key schedule defined in Section 2.

Note that the functions above perform more than just cryptographic operations. Additional operations include initialisation, copying between buffers, and general glue-code. We further discuss the overhead relative to the cryptographic operations for a subset of these functions.

Our experiments were performed between two Amazon Web Service (AWS) dedicated m5.large EC2 instances in two different availability zones (AZs) in the London Region. Each instance runs lLinux 4.14 with an Intel Xeon Platinum 8175M 2.5 Ghz CPU. We chose this relatively short distance between the initiator and responder to remain faithful to the practical restrictions on the deployment of Muckle. The QKD is an inherent part of the protocol, and deployment of a QKD network currently has a maximum distance of approximately 100km between nodes. For both metrics, the median over 100 samples is reported and each process is pinned to a single CPU.

To contrast running C-Muckle over this short distance with a more typical real-world setting, we performed the same experiment between two m5.large EC2 instances in two different regions, London and Paris, but only measuring the wall-time.

Wall-time Complete Execution: The complete execution time for a Muckle protocol run between two AZs is approximately 12.9ms. In comparison, the complete execution time between two regions is 26ms. The measurement scope is the execution of one entire stage of Muckle, including networking, initialising contexts, running clean up functions and executing general glue-code. By contrast, the round-trip-times for simple pings between two AZs and two regions were 0.745 ms and 8.224 ms.

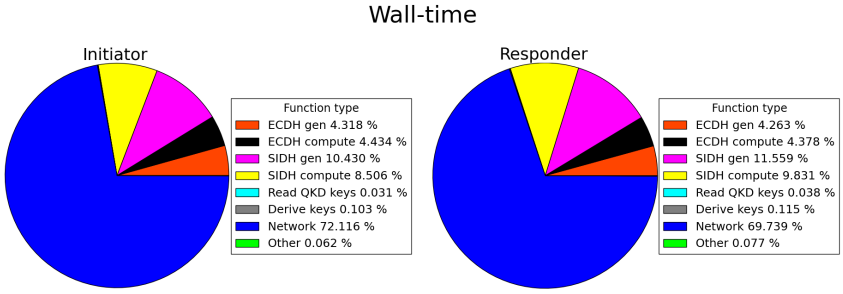

Fig. 2: Results of the wall-time measurement experiment between two AWS EC2 instances in two different regions (London and Paris). The top 6 categories for each chart are functions that correspond to C-Muckle functions described earlier. The network category includes time taken to initialise of the socket, as well as sending and receiving messages. The percentage for the Other category is computed by subtracting the median wall-time for the top 6 functions and the median time for networking from the entire median wall-time of the participant. (**Left**) C-Muckle initiator. (**Right**) C-Muckle responder.

Wall-time Function Profiling: Figure 3 in Appendix A provides a more granular view of the cost for specific functions in C-Muckle between two AZs. For the initiator, approximately 7.22ms is spent on various cryptographic function calls, with more than 68% of the 7.22ms spent performing SIDH-related operations. The same behaviour can be observed for the responder. The relative cost of cryptographic operations in C-Muckle is therefore more than 65% when it is run over the short distances between AZs.

Clock Cycle Function Profiling: Table 1 contains an overview of the measured number of clock cycles for various functions. Each cell contains two functions: the first in each cell is the C-Muckle function described above, while the second is the function from the library dependencies,[5] used to implement the cryptographic operations in the C-Muckle function, i.e. during the execution of e.g. muckle_ecdh_gen() the function mbedtls_ecdh_gen_public() from the mbedtls library will be called. The table highlights the absolute overhead of the cryptographic operations as implemented in C-Muckle compared to the core cryptographic operation supported via one of the two library dependencies. We have excluded the functions muckle_read_qkd_keys() and muckle_derive_keys() because their cost is negligible relative to the total cost of the execution flow in C-Muckle.

The overhead in cryptographic C-Muckle functions relative to the corresponding external library functions is less than 15,000 clock cycles with one spike at 77,000 clock cycles. The overhead is predominately from copying buffers, initialisation and retrieving parameters. The "compute" functions also involve key derivation steps.

[5] Either mbedtls or PQCrypto-SIDH

Function	Clock cycles
muckle_ecdh_gen()	2,769,893
mbedtls_ecdh_gen_public()	2,768,317
muckle_ecdh_compute()	2,875,367
mbedtls_ecdh_calc_secret()	2,846,614
initiator muckle_sidh_gen()	6,852,319
EphemeralKeyGeneration_A_SIDHp503()	6,775,268
initiator muckle_sidh_compute()	5,630,939
EphemeralSecretAgreement_A_SIDHp503()	5,613,257
responder muckle_sidh_gen()	7,531,586
EphemeralKeyGeneration_B_SIDHp503()	7,526,757
responder muckle_sidh_compute()	6,399,884
EphemeralSecretAgreement_B_SIDHp503()	6,391,934

Table 1: (**Left column**) The first function in each cell is a C-Muckle function described in the text. The second in each cell is the function from the library dependency, used to implement the C-Muckle function. The functions prefixed with mbedtls are from the mbedtls library, otherwise they are from the PQCrypto-SIDH library. (**Right column**) The median number of clock cycles over 100 samples.

The number of clock cycles for the ECDHE mbedtls functions using the elliptic curve curve25519, are far from state-of-the-art. For example, Bernstein [9] reports a total of 832,457 clock cycles for both key generation and secret key computation. It should therefore be possible to significantly improve the ECDHE performance in C-Muckle using a different library to mbedtls. However, we have found the mbedtls library to be easier to work with than other available libraries (like OpenSSL).

4 Hybrid Security Framework

Here we introduce our multi-stage hybrid authenticated key exchange (AKE) security framework HAKE for the analysis of our new protocol. HAKE follows the tradition of standard Bellare-Rogaway-based AKE models, and cleanly captures adversaries of differing strength (quantum and classical) via a fine-grained key compromise interface. Specifically, we model quantum adversaries by allowing them to corrupt non-post-quantum key exchange mechanisms (for instance, discrete logarithm-based key exchange algorithms). We highlight that our HAKE framework is flexible, and extends beyond Muckle, as HAKE captures (for example) the use of long-term asymmetric secrets, which are not used within Muckle. This allows HAKE to capture a variety of hybrid schemes, and is not simply to restricted to the use case of Muckle. We explain the HAKE framework in Section 4.2, and describe the corruption abilities of the adversary in Section 4.3. We then describe cleanness and partnering definitions in Sections 4.4 and 4.5.

4.1 Secret Key Generation

HAKE addresses secret key generation (the output of a "KeyGen" algorithm) of individual key exchange components explicitly, and categorises them into *long-term* (i.e. generated once and used in every execution of the protocol), and *ephemeral* (i.e. generated on a per-stage basis) secret generation. We further divide these into the following sub-categories:

Post-quantum asymmetric secret generation: The generation of a public-key pair for post-quantum code-based signature schemes is an example of a long-term variant. We denote the algorithm that generates these secrets as LQKeyGen. An algorithm that generates SIDH public-key pairs is an example of an ephemeral variant, which we denote as EQKeyGen.

Classical asymmetric secrets: An algorithm that generates long-term RSA public-key pairs for signatures (that do not offer post-quantum security), for example, would be denoted via LCKeyGen. Similarly, the generation of ECDHE public-key pairs would be done via ECKeyGen.

Symmetric secrets: Long-term preshared secret keys are generated via LSKeyGen, while (for instance), we consider that the ephemeral keying material generated by a quantum key distribution protocol to be captured as a ephemeral symmetric secret generation algorithm, which we denote ESKeyGen.

4.2 Execution Environment

Consider an experiment $\mathsf{Exp}^{\mathsf{HAKE},\mathsf{clean},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda)$ played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. $\mathcal{C}$ maintains a set of n_P parties $P_1,\ldots,P_{n_P}$ (representing users interacting with each other in protocol executions), each capable of running up to (potentially parallel) n_S sessions of a probabilistic key-exchange protocol Π. Each session can consist of up to n_T consecutive stages, each an execution of the key-exchange protocol Π, represented as a tuple of algorithms $\Pi = (f, \mathsf{EQKeyGen}, \mathsf{ECKeyGen}, \mathsf{ESKeyGen}, \mathsf{LQKeyGen}, \mathsf{LCKeyGen}, \mathsf{LSKeyGen})$. We use π_i^s to refer to both the identifier of the s-th instance of the Π being run by party P_i and the collection of per-session variables maintained for the s-th instance of Π run by P_i, and f is a algorithm capturing the honest execution of the protocol Π by protocol participants. We describe generically these algorithms below:

$\Pi.f(\lambda, \boldsymbol{pk}_i, \boldsymbol{sk}_i, \boldsymbol{pskid}_i, \boldsymbol{psk}_i, \pi, m) \xrightarrow{\$} (m', \pi')$ is a (potentially) probabilistic algorithm that takes a security parameter λ, the set of long-term asymmetric key pairs $\boldsymbol{pk}_i, \boldsymbol{sk}_i$ of the party P_i, a collection of per-session variables π and an arbitrary bit string $m \in \{0,1\}^* \cup \{\emptyset\}$. f outputs a response $m' \in \{0,1\}^* \cup \{\emptyset\}$ and an updated per-session state π', behaving as an honest protocol implementation.

We describe a set of algorithms $\Pi.\mathsf{XYKeyGen}(\lambda) \xrightarrow{\$} (pk, sk)$, where $X \in \{\mathsf{E},\mathsf{L}\}$ and $Y \in \{\mathsf{C},\mathsf{Q}\}$. $\Pi.\mathsf{XYKeyGen}$ is a probabilistic *post-quantum ephemeral* (if $\mathsf{XY} = \mathsf{EQ}$), *post-quantum long-term* (if $\mathsf{XY} = \mathsf{LQ}$), *classic ephemeral* (if $\mathsf{XY} = \mathsf{EC}$), or *classic long-term* (if $\mathsf{XY} = \mathsf{LC}$) asymmetric keygen algorithm, taking a security parameter λ and outputting a public-key/secret-key pair (pk, sk).

We describe a set of algorithms $\Pi.\mathsf{ZSKeyGen}(\lambda) \xrightarrow{\$} (psk, pskid)$, where $Z \in \{\mathsf{E},\mathsf{L}\}$. $\Pi.\mathsf{ZSKeyGen}$ is a probabilistic *ephemeral* (if $\mathsf{Z} = \mathsf{E}$), or *long-term* (if

$\mathsf{Z} = \mathsf{L}$) symmetric key generation algorithm taking as input a security parameter λ and outputting some symmetric keying material and (potentially) a keying material identifier $(psk, pskid)$, (or $(qkm, qkmid)$, respectively).

$\mathcal{C}$ runs $\Pi.\mathsf{LQKeyGen}(\lambda)$, $\Pi.\mathsf{LCKeyGen}(\lambda)$ and $\Pi.\mathsf{LSKeyGen}(\lambda)$ n_P times to generate asymmetric post-quantum and classical key pairs (which we denote with $\boldsymbol{pk}_i, \boldsymbol{sk}_i$) for each party $P_i \in \{P_1, \ldots, P_{n_P}\}$ as well as a symmetric keys and identifier $(\boldsymbol{psk}, \boldsymbol{pskid})$ and delivers all public-keys $\boldsymbol{pk}_i$, $\boldsymbol{pskid}$ for $i \in \{1, \ldots, n_P\}$ to $\mathcal{A}$. The challenger $\mathcal{C}$ then randomly samples a bit $b \xleftarrow{\$} \{0,1\}$ and interacts with the adversary via the queries listed in Section 4.3, also maintaing a set of corruption registers, containing a list of ephemeral and long-term secrets that have been compromised by $\mathcal{A}$ via Reveal, $\mathsf{Corrupt}$ and $\mathsf{Compromise}$ queries. Eventually, $\mathcal{A}$ terminates and outputs a guess d of the challenger bit b. The adversary wins the HAKE key-indistinguishability experiment if $d = b$, and additionally if the test session π satisfies a cleanness predicate clean, which we discuss in more detail in Section 4.5. We give an algorithmic description of this experiment in the full version. Each session maintains a set of per-session variables:

$\rho \in \{\texttt{init}, \texttt{resp}\}$**:** The role of the party in the current session. Note that parties can be directed to act as `init` or `resp` in concurrent or subsequent sessions.

$pid \in \{1, \ldots, n_P, \star\}$**:** The intended communication partner, represented with $\star$ if unspecified. Note that the identity of the partner session may be set during the protocol execution, in which case pid can be updated once.

$stid \in [n_T]$**:** The current (or most recently completed) stage of the session.

$\alpha \in \{\texttt{active}, \texttt{accept}, \texttt{reject}, \perp\}$**:** The status of the session, initialised with $\perp$.

$\mathbf{m}_i[stid] \in \{0,1\}^* \cup \{\perp\}$**, where** $\mathbf{i} \in \{\mathbf{s}, \mathbf{r}\}$**:** An array of the concatenation of messages sent (if $\mathbf{i} = \mathbf{s}$) or received (if $\mathbf{i} = \mathbf{r}$) by the session in each stage. Initialised by $\perp$ and indexed by the stage identifier $stid$.

$\mathbf{k}[stid] \in \{0,1\}^* \cup \{\perp\}$**:** An array of the session keys from each stage, or $\perp$ if no session key has yet been computed. Indexed by the stage identifier $stid$

$\mathbf{exk}[stid] \in \{0,1\}^* \cup \{\perp\}$**, where** $\mathbf{x} \in \{\mathbf{q}, \mathbf{c}, \mathbf{s}\}$**:** An array of the *post-quantum ephemeral asymmetric* (if $\mathbf{x} = \mathbf{q}$), *classic ephemeral asymmetric* (if $\mathbf{x} = \mathbf{c}$), or *ephemeral symmetric* (if $\mathbf{x} = \mathbf{s}$) secret keys used by the session in each stage. Initialised by $\perp$ and indexed by the stage identifier $stid$.

$\mathbf{pss}[stid] \in \{0,1\}^* \cup \{\perp\}$**:** Any per-stage secret state that is established during protocol execution for use in the following stage. Sessions use $\mathbf{pss}[stid - 1]$ during the protocol execution of stage $stid$. Indexed by $stid$.

$\mathbf{st}[stid] \in \{0,1\}^*$**:** Any additional state used by the session in each stage.

4.3 Adversarial Interaction

Our HAKE framework considers a traditional AKE adversary, in complete control of the communication network, able to modify, inject, delete or delay messages. They are able to compromise several layers of secrets: (a) long-term private keys, allowing our model to capture forward-secrecy notions and quantum adversaries. (b) ephemeral private keys, modelling the leakage of secrets due to the use of bad randomness generators, or potentially bad cryptographic primitives or quantum

adversaries. (c) preshared symmetric keys, modelling the leakage of shared secrets, potentially due to the misuse of the preshared secret by the partner, or the forced later revelation of these keys due to the compromise of partner devices. (d) ephemeral keying material, modelling attacks on the quantum key distribution. For instance, capturing things such as photon splitting attacks. (e) session keys, modelling the leakage of keys by their use in bad cryptographic algorithms. The adversary interacts with the challenger $\mathcal{C}$ via the queries below:

$\mathsf{Create}(i, j, role) \rightarrow \{(s), \perp\}$**:** Allows the adversary $\mathcal{A}$ to initialise a new session owned by party P_i, where the role of the new session is $role$, and intended communication partner party P_j. Note that if $\mathcal{A}$ has already initialised the intended partner session, $\mathcal{A}$ must give the session index r (indicating the intended partner session π_j^r) in order to synchronise ephemeral symmetric keys. If a session π_i^s has already been created, $\mathcal{C}$ returns $\perp$. Otherwise, $\mathcal{C}$ returns (s) to $\mathcal{A}$.

$\mathsf{Send}(i, s, m) \rightarrow \{m', \perp\}$**:** Allows $\mathcal{A}$ to send messages to sessions for protocol execution and receive the output. If the session $\pi_i^s.\alpha \neq \texttt{active}$, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ computes $\Pi.f(\lambda, \boldsymbol{pk}_i, \boldsymbol{sk}_i, \boldsymbol{pskid}_i, \boldsymbol{psk}_i, \pi_i^s, m) \rightarrow (m', \pi_i^{s\prime})$, sets $\pi_i^s \leftarrow \pi_i^{s\prime}$, updates transcripts $\pi_i^s.\mathbf{m}_r$, $\pi_i^s.\mathbf{m}_s$ and returns m' to $\mathcal{A}$.

$\mathsf{Reveal}(i, s, t)$**:** Allows $\mathcal{A}$ access to the session keys computed by a session. $\mathcal{C}$ checks if $\pi_i^s.\alpha[t] = \texttt{accept}$ and if so, returns $\pi_i^s.\mathbf{k}[t]$ to $\mathcal{A}$. In addition, the challenger checks if there exists another session π_j^r that *matches* with π_i^s, and also sets $\mathsf{SK}_j^r[t] \leftarrow \texttt{corrupt}$. Otherwise, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

$\mathsf{Test}(i, s, t) \rightarrow \{k_b, \perp\}$**:** Allows $\mathcal{A}$ access to a real-or-random session key k_b used in determining the success of $\mathcal{A}$ in the key-indistinguishability game. If a session π_i^s exists such that $\pi_i^s.\alpha = \texttt{accept}$, then the challenger $\mathcal{C}$ samples a key $k_0 \xleftarrow{\$} \mathcal{D}$ where $\mathcal{D}$ is the distribution of the session key, and sets $k_1 \leftarrow \pi_i^s.\mathbf{k}[t]$. $\mathcal{C}$ then returns k_b (where b is the random bit sampled during set-up) to $\mathcal{A}$. Otherwise $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

$\mathsf{CorruptXK}(\{i, j\}) \rightarrow \{k_i, \perp\}$**:** Allows $\mathcal{A}$ access to the secret preshared key $psk_i^j = psk_j^i$ (if $\mathsf{X} = \mathsf{S}$), the secret post-quantum long-term key qsk_i (if $\mathsf{X} = \mathsf{Q}$) or the secret classical long-term key csk_i (if $\mathsf{X} = \mathsf{C}$), generated for the party P_i (and P_j, in the preshared case) prior to protocol execution. If the secret long-term key has already been corrupted previously, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

$\mathsf{CompromiseYK}(i, s, t) \rightarrow \{\mathbf{esk}[t], \perp\}$**:** Allows $\mathcal{A}$ access to the secret ephemeral post-quantum key $\pi_i^s.\mathbf{eqk}[t]$ (if $\mathsf{Y} = \mathsf{Q}$), the secret ephemeral classical key $\pi_i^s.\mathbf{eck}[t]$ (if $\mathsf{Y} = \mathsf{C}$), or the secret ephemeral symmetric key $\pi_i^s.\mathbf{esk}[t]$ (if $\mathsf{Y} = \mathsf{S}$) generated for the session π_i^s prior to protocol execution in stage t. Note that if there exists another session π_j^r such that $\pi_i^s.\mathbf{esk}[t] = \pi_j^r.\mathbf{esk}[t]$, then that session's ephemeral symmetric key is also considered corrupted. If $\pi_i^s.\mathbf{esk}[t]$ has already been corrupted previously, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

$\mathsf{CompromiseSS}(i, s, t) \rightarrow \{\mathbf{pss}[t], \perp\}$**:** Allows the adversary access to the secret per-session state $\pi_i^s.\mathbf{pss}[t]$ generated by a session π_i^s during protocol execution. for use in the next stage of the session's protocol execution. Note that if there exists another session π_j^r such that $\pi_i^s.\mathbf{pss}[t] = \pi_j^r.\mathbf{pss}[t']$, then that

session's per-stage secret state is also considered corrupted. If $\pi_i^s.\mathbf{pss}[t]$ has already been corrupted previously, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

4.4 Partnering Definition

To evaluate the secrets that $\mathcal{A}$ can reveal without trivially breaking the security of the protocol, key-exchange models must first define how sessions are *partnered*. Otherwise, $\mathcal{A}$ would simply run a protocol between two sessions, faithfully delivering all messages, Test the first session to receive the real-or-random key, and Reveal the other session's key. If the keys are equal, then the Test key is real, and otherwise the session key has been sampled randomly.

In our work, we use both the matching definition *matching sessions* defined in the original eCK model [21], and *origin sessions*, introduced by Cremers and Feltz [17]. On a high level, π_i^s is an origin session of π_j^r if π_i^s has received the messages that π_j^r sent without modification, even if the reply that π_i^s sent back has not been received by π_j^r. If all messages sent and received by π_i^s and π_j^r are identical, then the sessions *match*.

4.5 Cleanness Predicates

We now define the exact combinations of secrets that an adversary $\mathcal{A}$ is allowed to compromise without trivially breaking a hybrid key exchange protocol. However, we note that the cleanness predicate defined below is specific to Muckle, and the threat model that Muckle intends to defend against. Other predicates, both stronger and weaker, can be constructed.

We wish to capture security against a quantum-equipped adversary, so a successful adversary is allowed to compromise the long-term and ephemeral classical asymmetric secrets without penalty. Since Muckle itself does not use public-key cryptography to authenticate its messages, we allow $\mathcal{A}$ to compromise the long-term asymmetric secrets (however, the challenger $\mathcal{C}$ will respond to CorruptCK and CorruptQK queries with $\perp$).

Since we wish to capture perfect forward secrecy, we allow a successful adversary to have issued a Test query to a session π_i^s owned by a party P_i (with no origin session) that has had its long-term symmetric key compromised previously, as long as the session was completed before the $\mathsf{CorruptSK}(i, j)$ query was issued (and $\pi_i^s.pid = j$). In addition, our construction should be post-compromise secure (as explored by Cohn-Gordon et al. [15]), so our cleanness predicate allows an adversary to have compromised *all* ephemeral secrets associated with a particular stage as long as there exists some stage previous that has not had all its ephemeral secrets compromised and the adversary has been passive in all stages between the "Test" stage and the previous "clean" stage.

Coming full circle then, a "clean" stage intuitively is one where the adversary has not compromised *all* of: (a) the ephemeral classic secrets of the Test session and its matching partner in the tested stage (b) the ephemeral post-quantum secrets of the Test session and its matching partner in the tested stage (c) the previous per-stage secrets shared by the Test session and its matching session in

the tested stage, and (d) the quantum keying material / ephemeral symmetric secrets shared by the Test session and its matching session in the tested stage.

It may also be desirable to determine the security guarantees that Muckle provides in the event of a new vulnerability discovered in the underlying post-quantum asymmetric key-exchange primitive, or a side-channel attack being discovered in the hardware of the QKD system. In order to capture this scenario, we provide a second cleanness predicate that captures non-quantum-equipped adversaries, which we denote $\mathsf{clean}_{c\mathsf{HAKE}}$. It is more-or-less identical to $\mathsf{clean}_{q\mathsf{HAKE}}$, but the adversary is allowed to compromise the ephemeral secrets (either symmetric or post-quantum) if they do not additionally compromise classic ephemeral secrets derived during the protocol execution. Now, we formalise the advantage of a QPT algorithm $\mathcal{A}$ in winning the HAKE key indistinguishability experiment in the following way:

Definition 1 (HAKE Key Indistinguishability). *Let Π be a key-exchange protocol, and n_P, n_S, $n_T \in \mathbb{N}$. For a particular given predicate* clean*, and a QPT algorithm $\mathcal{A}$, we define the advantage of $\mathcal{A}$ in the* HAKE *key-indistinguishability game to be* $\mathsf{Adv}^{\mathsf{HAKE},\mathsf{clean},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda) = 2 \cdot \left| \Pr\left[\mathsf{Exp}^{\mathsf{HAKE},\mathsf{clean},\mathcal{A}}_{\Pi,n_P,n_S n_T}(\lambda) = 1\right] - \frac{1}{2}\right|$*. We say that Π is post-quantum* HAKE*-secure if, for all $\mathcal{A}$,* $\mathsf{Adv}^{\mathsf{HAKE},\mathsf{clean},\mathcal{A}}_{\Pi,n_P,n_S,n_T}(\lambda)$ *is negligible in the security parameter λ.*

5 Security Analysis

This section is dedicated to presenting our main result Theorem 1. Due to space constraints, the full proof and advantage bound can be found in the full version. For our proof, we require post-quantum analogues of various security assumptions. These are mostly identical to classical prf, dual-prf ind-cpa and eufcma assumptions, but require security against all quantum probabilistic polynomial-time algorithms. It is also desirable to assess security of Muckle with respect to classic probabilistic polynomial-time adversaries, and thus we also prove that Muckle is HAKE-secure with cleanness predicate $\mathsf{clean}_{c\mathsf{HAKE}}$. Recall that $\mathsf{clean}_{c\mathsf{HAKE}}$ used here is a generalisation of $\mathsf{clean}_{q\mathsf{HAKE}}$, allowing us to establish key indistinguishability security in the scenario where the classical cryptographic component of Muckle remains secure and uncompromised, even if the post-quantum and QKD components both become insecure.

Theorem 1. *The* Muckle *key exchange protocol is* HAKE*-secure with cleanness predicate* $\mathsf{clean}_{q\mathsf{HAKE}}$ *(capturing perfect forward secrecy and post-compromise security) under the* qprf*,* eufqcma*,* dual-qprf*, and* qind-cpa *assumptions of* PRF*,* MAC*,* PRF *and* KEM*, respectively. That is, for any QPT algorithm $\mathcal{A}$ against the* HAKE *key-indistinguishability game* $\mathsf{Adv}^{\mathsf{HAKE},\mathsf{clean}_{q\mathsf{HAKE}},\mathcal{A}}_{\mathsf{Muckle},n_P,n_S,n_T}(\lambda)$ *is negligible in the security parameter λ.*

Proof (Sketch). We begin by dividing the proof into three separate cases where the query $\mathsf{Test}(i,s,t)$ has been issued: 1. The session π^s_i (where $\pi^s_i.\rho = \texttt{init}$)

has no origin session in stage t. 2. The session π_i^s (where $\pi_i^s.\rho = \texttt{resp}$) has no origin session in stage t. 3. The session π_i^s in stage t has a matching session.

We bound the probability of each case, and show that under certain assumptions, the advantage of $\mathcal{A}$ winning the key-indistinguishability game is negligible.

Cases 1 & 2: Test session without origin session We show that $\mathcal{A}$ has negligible change in causing π_i^s to reach an `accept` state without an origin session. By the cleanness predicate, if π_i^s `accept`s without an origin session, then $\mathcal{A}$ cannot have exposed the preshared symmetric key between π_i^s and its intended partner, before that point. We first replace the computation of $mkey_B$ with a uniformly random value $\widetilde{mkey_B}$ in stage t of π_i^s. Since $mkey_B = \mathsf{PRF}(PSK, SecState)$ (where $PSK = psk_i^j$ is itself uniformly random and independent), this is a sound replacement. Any $\mathcal{A}$ that can distinguish this change can be turned into an algorithm that breaks the post-quantum PRF assumption. Next, we define an abort event that triggers if π_i^s verifies a MAC tag in stage t successfully. We show that if $\pi_i^s.\alpha[t] \leftarrow \texttt{accept}$ without a matching session, that means that a MAC tag τ was produced that verified correctly, but there existed no matching session that produced τ. Any adversary capable of triggering this abort event can thus break the eufcma security of the MAC scheme.

Case 3: Test session with matching session Here, we show that if $\mathcal{A}$ that has issued a $\mathsf{Test}(i, s, t)$ query to a clean session π_i^s in stage t, then $\mathcal{A}$ has negligible advantage in guessing the test bit b. In what follows, we split our analysis of Case 3 into the following sub-cases, each corresponding to a condition necessary for the cleanness predicate to be upheld by π_i^s in stage t: 3.1 $\mathsf{CompromiseQK}(i, s, t)$, $\mathsf{CompromiseQK}(j, r, t)$ have not been issued, where π_j^r *matches* π_i^s in stage t; 3.2 $\mathsf{CompromiseSK}(i, s, t)$, $\mathsf{CompromiseSK}(j, r, t)$ have not been issued, where π_j^r *matches* π_i^s in stage t; 3.3 $\mathsf{CompromiseQK}(i, s, t')$, $\mathsf{CompromiseQK}(j, r, t')$ have not been issued, where π_j^r *matches* π_i^s in stages u such that $t' \leq u < t$ and no $\mathsf{CompromiseSS}(i, s, u)$, $\mathsf{CompromiseSS}(j, r, u)$ queries have been issued; 3.4 $\mathsf{CompromiseSK}(i, s, t')$, $\mathsf{CompromiseSK}(j, r, t')$ have not been issued, where π_j^r *matches* π_i^s in stages u such that $t' \leq u < t$ and no $\mathsf{CompromiseSS}(i, s, u)$, $\mathsf{CompromiseSS}(j, r, u)$ queries have been issued.

It is clear to see that the advantage of $\mathcal{A}$ in Case 3 is bound by the sum of the advantage of $\mathcal{A}$ in all subcases. Due to space restrictions, we only detail the proof sketch of Subcase 3.1. The other subcases follow similar proof strategies.

3.1: $\mathsf{CompromiseQK}(i, s, t)$, $\mathsf{CompromiseQK}(j, r, t)$ have not been issued, where π_j^r *matches* π_i^s in stage t. We begin by guessing the parties, sessions and stage such that $\mathcal{A}$ issues $\mathsf{Test}(i, s, t)$ and π_i^s matches π_j^r (aborting if our guess was incorrect). After, we replace the secret QKEM value qsk with the uniformly random and independent value $\widetilde{qsk}$, by interacting with a ind-cpa QKEM challenger and replacing the qpk_A, qpk_B values sent in m_0 and m_1 respectively with challenge values from the ind-cpa challenger. By the definition of this case, we know that the QKEM values exchanged in m_0 and m_1 were not modified be-

tween the sessions. Next, we replace the quantum key qk and the first chaining key k_0 (in the test session and its matching partner) with uniformly random values $\widetilde{qk}$, $\widetilde{k_0}$ from the same distribution. Since $qk = \mathsf{PRF}(\widetilde{qsk}, \mathsf{label}_{qk})$ and $k_0 \leftarrow \mathsf{PRF}(\widetilde{qk}, m_0 \| m_1)$ where $\widetilde{qsk}$, $\widetilde{qk}$ are uniformly random and independent, these are sound replacements. Any algorithm $\mathcal{A}$ that can distinguish these changes can be used to break the post-quantum PRF assumption.

In the next steps, we iteratively replace the second, third and fourth chaining keys $k_1 \leftarrow \mathsf{PRF}(ck, \widetilde{k_0})$, $k_2 = \mathsf{PRF}(qkm, \widetilde{k_1})$, and $k_3 = \mathsf{PRF}(SecState, \widetilde{k_2})$, with uniformly random values $\widetilde{k_1}$, $\widetilde{k_2}$, $\widetilde{k_3}$ from the same distribution. Unlike previous steps, we require a post-quantum $\mathsf{dual\text{-}prf}$ assumption here, but otherwise these changes proceed identically to the replacement of qsk and qk. Any algorithm $\mathcal{A}$ that can distinguish these changes can be used to break the post-quantum $\mathsf{dual\text{-}prf}$ assumption.

In this final step, we replace the secret state and session keys $SecState$, sk_A, $sk_B \leftarrow \mathsf{PRF}(\widetilde{k_3}, m_0 \| m_1 \| ctr)$ with uniformly random and independent values $\widetilde{SecState}, \widetilde{sk_A}, \widetilde{sk_B}$. As before, any algorithm $\mathcal{A}$ that can distinguish this change can be used to break the post-quantum PRF assumption.

Since $\widetilde{sk_A}, \widetilde{sk_B}$ are now uniformly random and independent values independent of the protocol flow regardless of the value of the test bit b, $\mathcal{A}$ has no advantage in guessing the test bit.

6 Future Work

Our paper opens up many avenues for future work. First, we have strongly abstracted the QKD component in our framework, treating is as an inexhaustible supply of shared, random bits. Yet there is a fine tradition of developing security proofs for QKD systems based purely on physical models. It is a significant challenge to integrate such approaches in our framework. The work of [24] provides one route forward using Universal Composability. In future QKD systems, the bit-rate of key agreement will exceed that which can be achieved by classical communication, at least over short ranges. This suggests adapting Muckle to allow rapid key refreshing from QKD keying material and slower refreshing from other sources. Our analysis framework could be extended to support such "differential refreshing". But this approach also raises implementation challenges, particularly around key synchronisation, which would need to be carefully addressed in order to avoid DoS and other attacks.

Acknowledgments

The research of Dowling was supported by Innovate UK and EPSRC grant EP/L018543/1 (the EQUIP project). The research of Hansen was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1). The research of Paterson was supported by Innovate UK and EPSRC grants EP/L018543/1, EP/K035584/1 and EP/M013472/1 and a gift from VMware.

Bibliography

[1] ARM mbed TLS. https://tls.mbed.org/. Accessed: 12-11-2018.

[2] C-Muckle source code. https://github.com/himsen/muckle. Accessed: 29-01-2020.

[3] Microsoft PQCrypto-SIDH. https://github.com/Microsoft/PQCrypto-SIDH. Accessed: 12-11-2018.

[4] M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. Estimate all the {LWE, NTRU} schemes! In D. Catalano and R. D. Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 351–367. Springer, 2018.

[5] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In T. Holz and S. Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343. USENIX Association, 2016.

[6] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, Heidelberg, Aug. 1996.

[7] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, Aug. 1994.

[8] C. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, 175(P1), 1984.

[9] D. J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, Apr. 2006.

[10] D. J. Bernstein. Is the security of quantum cryptography guaranteed by the laws of physics? *CoRR*, abs/1803.04520, 2018.

[11] N. Bindel, J. Brendel, M. Fischlin, B. Goncalves, and D. Stebila. Hybrid key encapsulation mechanisms and authenticated key exchange. In *International Conference on Post-Quantum Cryptography*, pages 206–226. Springer, 2019.

[12] N. Bindel, U. Herath, M. McKague, and D. Stebila. Transitioning to a quantum-resistant public key infrastructure. In T. Lange and T. Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 384–405. Springer, 2017.

[13] M. Braithwaite. Experimenting with post-quantum cryptography. Retrieved from URL, https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html, July 2016.

[14] J. Brendel, M. Fischlin, and F. Günther. Breakdown Resilience of Key Exchange Protocols: NewHope, TLS 1.3, and Hybrids. In K. Sako, S. Schneider, and P. Y. A. Ryan, editors, *Computer Security – ESORICS 2019*, pages 521–541, Cham, 2019. Springer International Publishing.

[15] K. Cohn-Gordon, C. J. F. Cremers, and L. Garratt. On Post-compromise Security. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 164–178. IEEE Computer Society, 2016.

[16] C. Costello, P. Longa, and M. Naehrig. Efficient algorithms for supersingular isogeny diffie-hellman. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 572–601. Springer, Heidelberg, Aug. 2016.

[17] C. J. F. Cremers and M. Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In S. Foresti, M. Yung, and F. Martinelli, editors, *ESORICS 2012*, volume 7459 of *LNCS*, pages 734–751. Springer, Heidelberg, Sept. 2012.

[18] A. Huang, S.-H. Sun, Z. Liu, and V. Makarov. Quantum key distribution with distinguishable decoy states. *Phys. Rev. A*, 98:012330, 2018.

[19] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Heidelberg, Aug. 2010.

[20] K. Kwiatkowski and L. Valenta. The tls post-quantum experiment. Retrieved from URL, `https://blog.cloudflare.com/the-tls-post-quantum-experiment`, October 2010.

[21] J. Li, K. Kim, F. Zhang, and X. Chen. Aggregate proxy signature and verifiably encrypted proxy signature. In W. Susilo, J. K. Liu, and Y. Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 208–217. Springer, Heidelberg, Nov. 2007.

[22] D. Moody. What was NIST thinking? Round 2 of the NIST PQC "Competition". Talk at Oxford University, 2019.

[23] M. Mosca, D. Stebila, and B. Ustaoglu. Quantum key distribution in the classical authenticated key exchange framework. In P. Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, volume 7932 of *Lecture Notes in Computer Science*, pages 136–154. Springer, 2013.

[24] J. Müller-Quade and R. Renner. Composability in quantum cryptography. *CoRR*, abs/1006.2215, 2010.

[25] J. Schank and D. Stebila. A Transport Layer Security (TLS) Extension For Establishing An Additional Shared Secret. IETF Draft, 2017.

[26] P. Sibson, C. Erven, M. Godfrey, S. Miki, T. Yamashita, M. Fujiwara, M. Sasaki, H. Terai, M. G. Tanner, C. M. Natarajan, R. H. Hadfield, J. L. O'Brien, and M. G. Thompson. Chip-based quantum key distribution. *Nature Communications*, 8:13984, Feb. 2017.

[27] D. Stebila, S. Fluhrer, and S. Gueron. Design issues for hybrid key exchange in TLS 1.3. IETF Draft, `https://tools.ietf.org/id/draft-stebila-tls-hybrid-design-01.html0`, 2019.

[28] A. Vakhitov, V. Makarov, and D. R. Hjelme. Large pulse attack as a method of conventional optical eavesdropping in quantum cryptography. *J. Mod. Opt.*, 48:2023, 2001.

[29] W. Whyte, S. Fluhrer, Z. Zhang, and O. Garcia-Morchon. Quantum-safe hybrid (QSH) key exchange for transport layer security (TLS) version 1.3. IETF Draft, 2017.

[30] H. P. Yuen. Security of quantum key distribution. *IEEE Access*, 4:724–749, 2016.

[31] R. Zhang, G. Hanaoka, J. Shikata, and H. Imai. On the security of multiple encryption or CCA-security+CCA-security=CCA-security? In F. Bao, R. Deng, and J. Zhou, editors, *PKC 2004*, volume 2947 of *LNCS*, pages 360–374. Springer, Heidelberg, Mar. 2004.

A Wall-time Function Profiling in Two Availability Zones

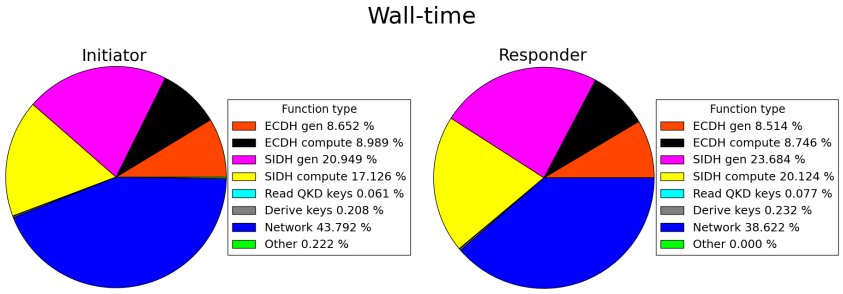

Fig. 3: Results of the wall-time measurement experiment between two AWS EC2 instances in two different availability zones located in the same region (London). Specifically, the chart captures the relative median wall-time spent executing various functions in the C-Muckle execution flow. The top 6 categories for each chart are functions that correspond to C-Muckle functions described in the text. The network category includes time taken to intialise of the socket, as well as sending and receiving messages. The percentage for the Other category is computed by subtracting the median wall-time for the top 6 functions and the median time for networking from the entire median wall-time of the participant. (**Left**) C-Muckle initiator. (**Right**) C-Muckle responder.