A Formal Analysis of the iMessage PQ3 Messaging Protocol

Technical Report

David Basin, Felix Linker, Ralf Sasse

Abstract

We report on the design and verification of a highly performant, device-to-device messaging protocol offering strong security guarantees even against an adversary with quantum computing capabilities, called iMessage PQ3. The protocol leverages Apple's identity services together with a custom, post-quantum secure initialization phase and afterwards it employs constructs from a double ratchet in the style of Signal, extended to provide post-quantum, post-compromise security. We present a detailed formal model of the protocol, a precise specification of its fine-grained security properties, and machine-checked proofs using the Tamarin prover. Particularly novel are the integration of post-quantum secure key encapsulation into the relevant protocol phases and the detailed security claims along with their complete formal analysis, covering both key ratchets, including unbounded loops.

1 Introduction

Research on secure instant messaging goes back over two decades, with early proposals including Off-the-Record Messaging, the Silent Circle Instant Messaging Protocol, iMessage, and Signal. Over time, the security community's understanding of the threat models and security claims for secure messaging evolved. Modern messaging protocols now offer very strong guarantees and can communicate messages secretly even in the presence of adversaries who can compromise different parties in different ways during the protocol's execution. This is befitting given that strong adversaries, like nation states, are capable of compromising both messaging servers and the end points sending and receiving messages, or compelling their release of information. More recently, security against adversaries with quantum computing capabilities has also become an important concern. This requires protection against adversaries who can “harvest now and decrypt later” by storing the encrypted data they intercept and decrypting it in the future when quantum computers become sufficiently powerful.

In this paper, we present and analyze an advanced secure messaging protocol, the iMessage PQ3 Messaging Protocol (PQ3). This protocol is a device-to-device protocol that will be used in production across all of Apple’s devices for device-to-device messaging using Apple’s Identity Services, which in turn underlies many other Apple services including iMessage, FaceTime, HomeKit, and HomePod hand-off. The protocol is designed to be performant and to offer strong guarantees against powerful adversaries, including those that may possess quantum computers in the future. Roughly speaking, the protocol employs a custom post-quantum secure initialization phase followed by elements of a double-ratchet construction, similar to that used by Signal [4]. Both phases combine classical cryptographic primitives, like elliptic curve Diffie-Hellman, and post-quantum primitives, namely Kyber, a module-lattice-based key-encapsulation mechanism.

We have carried out a detailed analysis of PQ3’s design using the Tamarin prover [3,5], a state-of-the art security protocol model checker. We report here on our model of this protocol, the adversary assumptions, and the protocol’s desired properties. In more detail, we use Tamarin’s expressive specification language to specify the messaging protocol and its use of both classical and post-quantum cryptography. We also specify all forms of adversary compromise; essentially the adversary can compromise any key at any time. We also use Tamarin’s property language to formalize and prove fine-grained secrecy and authenticity theorems. These theorems express the protocol’s security, under precise, fine-grained conditions capturing many desirable security guarantees.

Our analysis establishes that PQ3 provides strong security guarantees against an active network adversary that can compromise all secret keys unless explicitly stated otherwise. For example, PQ3 provides forward secrecy, post-compromise security, and post-quantum security with respect to a harvest now, decrypt later adversary. Notably, and in contrast to Signal, PQ3 provides post-compromise security also against active adversaries and not only against passive adversaries.

Contributions. Our main contributions are twofold. First, we formalize PQ3’s security guarantees and use Tamarin to provide formal, machine-checked proofs that PQ3 offers strong security guarantees against a powerful adversary with quantum computing capabilities. PQ3 thereby meets the expectations of a modern device-to-device messaging system. This high assurance is important given the prominent role of this protocol, which will be used on billions of devices worldwide.

Second, we show that symbolic security protocol model checkers, in particular Tamarin, can handle substantial, real-world protocols with nested loops, in their full complexity. This is nontrivial as it entails reasoning about unboundedly many instances of the protocol running in parallel, where the runs themselves are unbounded. In fact, it was commonly believed that "unbounded (looping) protocols like Signal, and protocols with mutable recursive data structures [...] are also out of scope for symbolic provers, without introducing artificial restrictions [2]." Our verification shows that this is not the case.

Organization. We begin by covering PQ3's requirements and threat model in Section 2. Afterwards, in Section 3 we describe the PQ3 Messaging Protocol. In Section 4 we describe our Tamarin model of the protocol as well as the adversary model, protocol properties, and some details on our proofs. In Section 5 we draw conclusions.

2 Threat Model and Protocol Requirements

2.1 Security Requirements

The two central security properties for PQ3 concern message secrecy and authenticity.

Secrecy. PQ3 was designed to provide strong secrecy guarantees, namely *message secrecy*, *forward secrecy*, and *post-compromise security* (also sometimes called self-healing or backward secrecy). Message secrecy means that as long as neither participants' session states are revealed, the adversary cannot learn any of their exchanged messages. Forward secrecy protects protocol participants against the future compromise of past sessions, e.g., the loss of a long-term secret later should not jeopardize the secrecy of previously exchanged messages. Post-compromise security helps to recover or "self heal" from a past compromise to communicate secretly again in the present and future.

We deliberately describe PQ3's secrecy guarantees first at a high level and note that PQ3 provides much finer grained notions of secrecy. In our security analysis of PQ3, we define a secrecy lemma that captures all three notions of secrecy and that considers the precise implications of partial session state compromise. Describing this fine-grained secrecy lemma though requires a detailed understanding of the key material used in PQ3, and is thus left for later.

Authentication. A message recipient can identify the message's sender. We can formulate this as an *agreement property*: the recipient's view of the message (including the identify of its peer, the sender), agree with the peer's view. This means that for any message received, allegedly originating from the peer at message counter i, that the peer has actually sent the message using counter i, and intending it to go to the receiver.

Replay Protection. While secrecy and agreement prevent an adversary from learning the contents of messages and faking a message's sender, they might still be able to trick a recipient into accepting an honestly sent message multiple times. PQ3 aims to provide cryptographic *replay protection*, which prevents such attacks.

2.2 Threat Model

PQ3 seeks to provide all of the above security properties against a strong active network adversary who will have access to a powerful quantum computer in the future. As an active network adversary, the adversary can read, reorder, intercept, replay, and send any message to any participant. As is standard in the symbolic setting, we assume that devices use strong randomness and that the adversary cannot, in general, factor large numbers or compute discrete logs and therefore cryptographic primitives like (elliptic curve) Diffie-Hellman are secure against this adversary. They can only decrypt ciphertexts if they possesses the corresponding secret keys. An exception to this are the adversary's quantum computing capabilities, which we describe next.

We assume that the adversary does *not yet* possess a quantum computer. The adversary anticipates developments in quantum computing and thus stores all messages sent. Therefore, they may be able break cryptographic primitives later, should sufficiently powerful quantum computers become available (also called "harvest now, decrypt later"). The only limitation we place on this adversary is that they cannot decapsulate secrets that were encapsulated with a post-quantum secure KEM like Kyber. But all other mechanisms, like (elliptic-curve) Diffie-Hellman, can be broken at that point.

The adversary can additionally, however, compromise any key at any time; hence our security properties must account for this. When we analyze our fine-grained security properties, we will consider the compromise of secret keys individually. This reflects that some keys may be in main memory and relatively easy to compromise, whereas others may be stored in the Apple Secure Enclave and are thus much harder to compromise.

Finally, we assume that PQ3's key directory, Apple's IDentity Services (IDS), is secure in that it only distributes authentic keys. The problem of key authentication is orthogonal to PQ3 and has recently been addressed by Apple with their plans for "Contact Key Verification" [1].

3 PQ3 Messaging Protocol

PQ3 is a device-to-device messaging protocol where either device can asynchronously exchange messages at any time, independent of the connection status of their peer's clients. For setup and session establishment, the protocol leverages the IDS key directory. We first describe PQ3 at a high-level of abstraction, followed by a more detailed, lower-level account. At the end of this section, we describe two additional performance-related design aspects of PQ3.

3.1 A High-level Account

If Alice wants to initiate messaging with Bob, she and Bob proceed as follows.

1. Alice queries the IDS to obtain *pre-key material* and a *long-term identity public key* for Bob.

2. Alice derives an initial *root key* and *message key.* Alice encrypts her first message for Bob using this message key and sends the ciphertext alongside a signature and the key material necessary to derive the initial root key to Bob.

3. Upon receiving this new message, Bob lacks the key to decrypt the ciphertext, so he must derive this. Bob first queries the IDS to verify Alice's long-term identity public key and checks the received signature. He then uses the key material just received to derive the initial root and message keys and decrypts the initial message. Alice and Bob have now established a shared session.

4. As long as the session does not change direction (i.e., the current sender keeps sending messages; initially, this will be Alice sending messages to Bob), both parties perform *symmetric ratcheting* to derive new message keys.

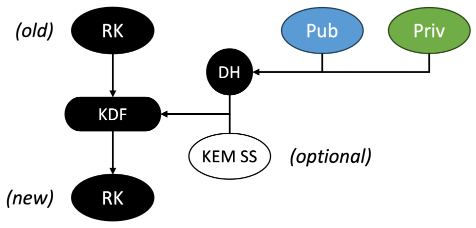

Figure 1: Sources of entropy for PQ3 root key derivation.

5. Whenever the session changes direction (i.e., the current receiver wants to reply; initially this will be Bob sending his first response to Alice) both parties perform a *public key ratchet*.

Note that at this high level of abstraction, Steps 2–5 resemble the standard double-ratchet construction. But there are significant differences in the concrete details on how the ratchets are performed, in particular how a post-quantum KEM is integrated into the ratcheting.

3.2 A More Detailed Account

We now explain PQ3 in more detail. This covers the keys used, how the root key is derived, how the session key is initially established, and how both the symmetric and the public key ratchets work. Although this account is more detailed than the previous one, we still focus on the essential ideas and we omit some low-level details concerning message and key derivation tags. Moreover, we describe some additional features of PQ3 at the end of this section.

PQ3 Keys Root keys are derived using three sources of entropy: (1) The previous root key, (2) an optional KEM shared secret, and (3) a shared ECDH secret. As usual, the ECDH shared secret is computed by combining the messaging partner's public key and one's own private key. Figure 1 illustrates the root key derivation. Furthermore, PQ3 messages are signed with the sender's long-term identity private key for authentication purposes. This necessitates that agents use three types of key pairs when running PQ3:

Long-term Identity Keys A P-256 ECDSA public/private key pair used for signatures for message authentication.

ECDH Keys A P-256 ECDH public/private key pair used to establish shared secrets.

KEM Keys A Kyber 768 or 1024 public/private key pair used to establish shared post-quantum-secure secrets.

To support asynchronous messaging, the agents control two types of ECDH and KEM keys. First, they upload ECDH and KEM *pre-keys* in timestamped pre-key bundles to the IDS. They sign these bundles with their long-term identity private key. The pre-keys enable agents to start new sessions without contacting their messaging partner first and are only used at the start of each session. Second, they control session-specific *ephemeral* versions of these keys. Participants will use Kyber 1024 keys for their KEM pre-keys and Kyber 768 keys for their ephemeral KEM keys.

Session Establishment In the following, we assume, as before, that Alice wishes to establish a new session with Bob. We depict an example run of PQ3, specifically showing the key derivations of both parties, in Figure 2. Alice initiates her session with Bob by first performing an IDS query with Bob's identity. Alice thereby learns three keys from the query's result: Bob's long-term identity public key, a ECDH public pre-key, and a KEM public pre-key; this is depicted within the white box in Figure 2, however that diagram omits showing the identity key as it is not used in any key derivations. Alice then generates a fresh ECDH ephemeral public/private key pair and samples a fresh KEM shared secret. Alice combines her ECDH ephemeral private key with Bob's ECDH public pre-key to obtain the initial ECDH shared secret, and she encapsulates the KEM shared secret she sampled with Bob's KEM public pre-key.

Alice now proceeds to derive the initial root key (and the associated initial chain key) from a zero-byte sequence, which stands in for the previous root key, together with the ECDH shared secret and the KEM shared secret, visible on the far left of Figure 2 in Step 1. She uses the initial chain key to derive a message key, which she uses to encrypt her initial message. She then sends Bob the following: The ciphertext, her ECDH ephemeral public key, the KEM encapsulation (with the latter two seen in Figure 2), a hash of Bob's pre-keys used (called the *pre-key hash*), and her signature on all of the above.

Bob can use that message to derive his view on the initial root key and associated chain key as follows. Bob first performs an IDS query to receive Alice's long-term identity public key (not depicted in Figure 2), which he uses to verify the message signature. The following steps are illustrated in the block numbered 1 in Figure 2. Bob then looks up the private parts of his pre-keys used by Alice using the pre-key hash. Bob decapsulates the

KEM encapsulation to obtain the KEM shared secret, and combines Alice's ECDH public ephemeral key with his ECDH private pre-key to establish the ECDH shared secret. With these two values (and the zero-byte sequence), Bob can now compute the initial root key and associated chain key, and derive a message key from that chain key, and decrypt the ciphertext.

Symmetric Ratchet With a shared root key established, Alice can send any number of additional messages to Bob without the participants updating the root key. Nevertheless, each of these messages will be encrypted with a distinct key that is derived by *symmetric ratcheting*. Whenever participants establish a new root key, they first derive an initial *chain key* in that root key derivation. Whenever a participant encrypts a message, they use the current chain key to derive a *message key*, and then ratchet the chain key forward by deriving a new chain key from the previous chain key. PQ3 establishes per-message forward secrecy as soon as the previous chain and message keys are deleted, i.e., participants should only store the latest root and chain key. The symmetric ratchet, though, is only executed as long as the conversation's direction does not change, i.e., as long as the current sender keeps sending. Whenever the current receiver wishes to send a message, a public key ratchet is performed instead, which we describe next.

Public Key Ratchet Suppose that, after receiving some messages from Alice, that Bob wants to reply. To do so, Bob first generates a fresh ECDH ephemeral public/private key pair. Depending on the conversation's state, Bob may additionally perform one of two actions:

(i) sample and encapsulate a new KEM shared secret, or

(ii) generate a new KEM ephemeral public/private key pair.

Action (i) is performed whenever Bob's peer, Alice, performed Action (ii) in the previous public key ratchet. Action (ii) need not always be performed and its application is determined by a custom heuristic. To save bandwidth, PQ3 does not require that KEM keys are refreshed with every public key ratchet.

Bob then derives the next root key and associated initial chain key. He first combines his freshly generated ECDH ephemeral private key with Alice's most recent ECDH ephemeral public key to obtain the new ECDH shared secret. He then uses the previous root key, the new ECDH shared secret, and either the new KEM shared secret or a zero-byte sequence (depending on whether Bob performed Action (i)) to derive the next root key and associated

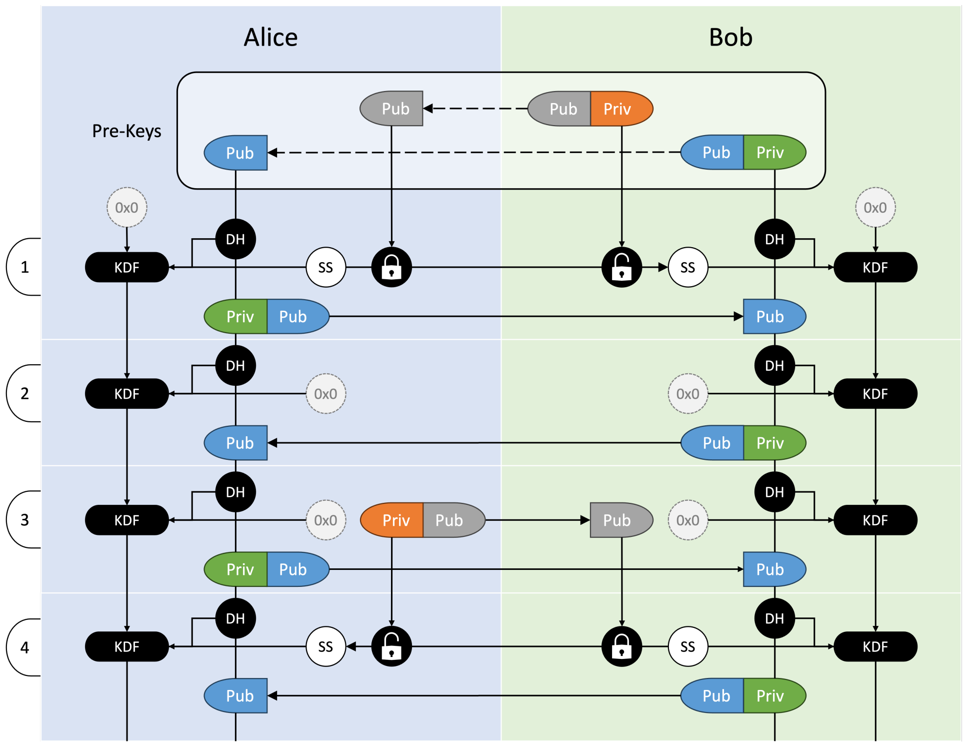

Figure 2: Illustration of PQ3's root key derivation. Alice initiates a session with Bob and initially uses pre-key material to derive a root key. Alice sends a freshly encapsulated shared KEM secret, and a freshly sampled ECDH public key to Bob that Bob can later use to derive session keys when he becomes active. New KEM shared secrets are only sampled and shared when the other participant sent a new KEM public key in the previous public key ratchet. Orange/gray key pairs denote Kyber KEM keys, and green/blue key pairs denote ECDH keys. The design of this figure was inspired by [4].

initial chain key. He again derives a message key from that chain key to encrypt his message and sends Alice the following values: the ciphertext, his fresh ECDH ephemeral public key, optionally the new KEM encapsulation (Action (i)), optionally his new KEM public key (Action (ii)), and a signature on all of the above.

Figure 2 depicts examples of public key ratcheting in Steps 2–4. In Step 3, Alice decides to generate a new ephemeral KEM public/private key pair and sends the corresponding public key to Bob, i.e., Alice executes Action (ii) above. This means that Bob will execute Action (i) in Step 4 of Figure 2.

3.3 Other Features of PQ3

PQ3 includes an implementation optimization and a fault tolerance mechanism that we omitted in our discussion for the sake of brevity and readability. First, whenever a participant switches from being the sender to being the receiver, they immediately perform both the public key ratchet to become receiver now and to become sender later. This is possible because when becoming receiver, the participant already has all information from their peer necessary for the public key ratchet step to become sender later. The receiver only needs to sample a fresh ephemeral ECDH private key in advance, store it, and send it to their peer when they later become the sender.

Second, for robustness, e.g., to tolerate out-of-order delivered messages, PQ3 clients maintain multiple so-called *incoming chains*. Should a client receive a message that skips a message counter, it saves the session state of the skipped counter in such an incoming chain. PQ3 allows receiving messages on the three most recent incoming chains, dropping the oldest whenever a client exceeds the limit.

Finally, we note that PQ3 requires clients to rotate their pre-keys stored on the IDS roughly every two weeks.

4 Tamarin Model

Tamarin is a state-of-the-art security protocol analysis tool, described in detail in [3, 5]. Here we just provide a high-level overview. We highlight that Tamarin works in the *symbolic model* of cryptography, where cryptographic primitives are assumed perfect, as previously stated. This allows Tamarin to be highly automated and to mechanically check all proofs.

4.1 Background on Tamarin

Tamarin uses labeled *multiset rewriting rules* to model the behavior of protocol participants. The participants play in so-called roles, where the possible actions of each role is given by sets of rules. Tamarin verifies security properties against an active network adversary who can read, intercept, reorder, replay, and send any message. In addition to this built-in adversary, modelers can give the adversary additional capabilities using explicit rules. Furthermore, setup assumptions can also be modeled with further rules.

Each rule consists of a premise and conclusion. These are made up of (potentially *persistent*) *facts*, which store the terms that the symbolic model reasons about. The state of the analysis of a protocol in Tamarin includes the states of all roles, the adversary knowledge, all exchanged messages, and more. To apply a rule, the facts in its premise must be found in the current state. When a rule is applied, all non-persistent facts appearing in the premise of this rule are removed from the state and replaced by instances of the facts in the conclusion of this rule.

To model different cryptographic primitives, Tamarin supports a number of built-in equational theories. These theories contain the necessary identities, to, e.g., allow the decryption of an encrypted message with the appropriate key. The user can additionally define their own equational theories under some technical conditions.

4.2 Protocol Model

We used Tamarin to comprehensively model PQ3, and an overview of our model is depicted in Figure 3. Our formal model centers around `Session` facts, which store all information necessary to send and receive messages. For example, a `Session` fact stores a participant's most recently generated ECDH and KEM private key and the corresponding public keys of their peer, as well as any derived root, message, and chain keys.

Our formal model has three parts. The first part models the generation of long-term signing keys and pre-keys (rule `UserKeyGen`), and IDS queries (rule `QueryIDS`). These are setup rules, which are the same for all participants, independent of whether they start as sender or receiver initially. The second part is the model of PQ3 itself, which we describe shortly. The third part models the adversary capabilities. We allow the adversary to compromise every private key through dedicated reveal-rules unless our security lemmas explicitly forbid a certain key to be revealed. Additionally, we model the harvest now, decrypt later adversary as follows. Whenever participants

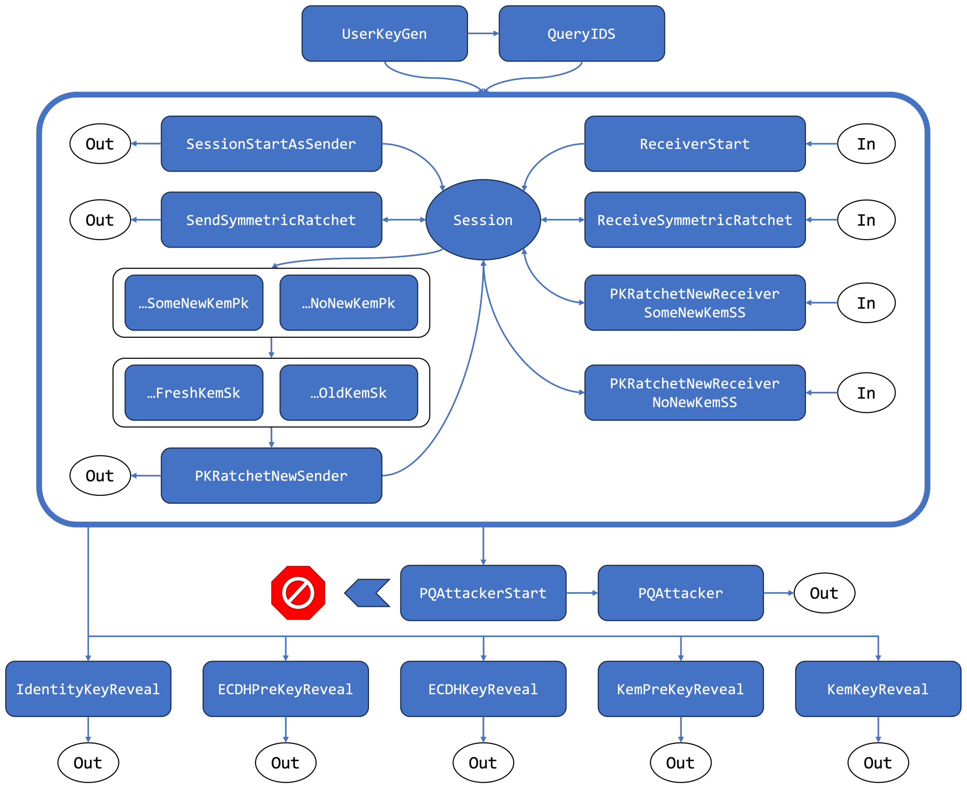

Figure 3: Overview of our formal model. Rectangles denote rules and ellipses denote facts, with their respective name printed inside. Arrows denote fact consumption and generation or rule transition. White rectangles denote a branch of two rules. The rule `PQAttackerStart` can be applied at any point. When this happens, protocol execution halts (modelling a harvest now, decrypt later adversary) and from then on the rule `PQAttacker` can be applied which reveals any non-post-quantum-secure secret to the adversary. Note that this figure names all multiset-rewriting rules used in our formal model.

generate a non-post-quantum-secure key, e.g., a fresh ephemeral ECDH private key, our model saves these keys in persistent facts (i.e., facts that are not consumed upon use in a rule's premise). The adversary can access any secrets stored this way after the rule `PQAttackerStart` was applied, but from that point on, no honest participant runs PQ3.

The model of PQ3's core is depicted in the center of Figure 3. The left-hand side shows all sender-related rule names and the right-hand all receiver related rule names. There are two rules to start a session as sender or receiver respectively (`SessionStartAsSender`/`ReceiverStart`). These are the only two rules that generate a new `Session` fact, and are only applied once per session. All other rules may be applied multiple times per session. After a new session has started, there are two options. Either, the conversation does not change direction, then both participants will apply the symmetric ratchet rules (`SendSymmetricRatchet`/`ReceiveSymmetricRatchet`), or the conversation changes direction and the public key ratchet rules are applied.

When a participant becomes the sender (after being the receiver), they first check whether a new KEM public key was sent in the previous public key ratchet and they afterwards generate a new KEM shared secret accordingly (`SomeNewKemPk`/`NoNewKemPk`). Then, they non-deterministically decide whether they want to generate a fresh KEM private key and send the corresponding public key (`FreshKemSk`/`OldKemSk`). Finally, they perform the public key ratchet and send the next message (`PKRatchetNewSender`).

When one becomes the receiver, one of two rules apply. In both rules, the change to receiver is triggered by receiving a new message while being in the sender state, which makes the participant perform the public key ratchet. In one of the two rules, they do so using a decapsulated KEM shared secret (`PKRatchetNewReceiverSomeNewKemSS`), and in the other rule they use a zero-bytes sequence instead (`PKRatchetNewReceiverNoNewKemSS`). Unlike our modelling of branches on the sender's side, we duplicate the public key ratchet rule for the receiver to simplify proofs. Note that the new receiver might receive a new KEM public key as well. However, they will only branch on this when they become sender again.

At a high level, one can consider our model as implementing two nested loops. First, there is the public key ratchet loop in which participants generate new ephemeral ECDH secret keys and generate new root keys. Second, there is the symmetric ratchet loop in which participants use a fixed root key to send multiple messages. The symmetric ratchet loop always runs within one public key ratchet loop step. To complete our description of the formal model, Figure 4 provides an example of a multiset-rewriting rule.

```
rule SessionStartAsASender:
  let dst          = <$Me, pk(~idKey), $Them, peerIdPk>
      ecdhSS       = peerEcdhPk^~ecdhSk
      kemEncap     = encap(~kemSS, peerKemPk)
      rkCK         = hkdf(hkdf(hkdf(ecdhSS, '0'), ~kemSS), dst)
      chainKey     = suffix(rkCK)
      preKeyHash   = h(<peerEcdhPk, peerKemPk, preKeySig>)
      msgKey       = hkdf(chainKey, 'msg_key')
      msgKeyInd    = hkdf(msgKey, 'msg_key_ind')
      nextChainKey = hkdf(chainKey, 'chain_key')
      ciphertext   = senc(~msg, msgKey)
      body         =
        <'msg_sig', ciphertext, msgKeyInd, 'g'^~ecdhSk, %1,
          dst, None, Just(kemEncap), Just(preKeyHash)>
      signature    = sign(body, ~idKey)
  in
  [ Fr(~id)
  , !IdentityKey($Me, ~idKey)
  , !KemPreKey($Me, ~kyberSk)
  , IDSQuery($Them, peerIdPk, peerEcdhPk, peerKemPk, preKeySig)
  , Fr(~msg), Fr(~ecdhSk), Fr(~kemSS) ]
  --[ /* omitted */ ]->
  [ Out(<ciphertext, signature, 'g'^~ecdhSk, %1, msgKeyInd,
         None, Just(kemEncap), Just(preKeyHash)>)
  , Session(~id, %1, 'S', $Me, ~ecdhSk, ~kyberSk, $Them,
      peerIdPk, peerEcdhPk, Just(peerKemPk), None, ~kemSS,
      peerKemPk, rkCK, chainKey)
  , Out('g'^~ecdhSk)
  , !NonPQSecKey(~ecdhSk)
  , !ECDHKey(~id, $Me, ~ecdhSk) ]
```

Figure 4: The Tamarin multiset-rewriting rule modelling the start of a new session as sender. We omitted action facts (transition labels) for clarity. In this rule, the sender receives their peer's long-term identity and pre-keys from the IDS (`IDSQuery`), looks up their own long-term identity and KEM pre-key (`!IdentityKey/KemPreKey`), samples a fresh KEM shared secret and ephemeral ECDH private key, and generates a session ID (`Fr`). They use these values to send the first message to their peer (`Out`) and save the session state (`Session`). Furthermore, they share their ephemeral ECDH public key with the adversary, and flag the ephemeral ECDH private key as non-post-quantum secure, as well as ECDH key to make it available to the adversary rules.

```
functions: hkdf/2, suffix/1, h/1

functions: pqpk/1, encap/2, decap/2
equations: decap(encap(k, pqpk(sk)), sk) = k

functions: default/2, Just/1, None/0, unjust/1
equations: default(Just(v), t) = v,
           default(None, v) = v,
           unjust(Just(t)) = t
```

Figure 5: Custom functions and equations defined in our formal model.

4.2.1 Signature and Equations for Base Model

We use Tamarin's built-in equational theories for signing, symmetric encryption, and Diffie-Hellman. These respectively model digital signatures, symmetric encryption under message keys, and ECDH key exchanges. We additionally use Tamarin's natural numbers theory to model message counters.

In addition to these built-in theories, we use some custom functions and equations, shown in Figure 5. First, we use the functions `hkdf` and `suffix` for key derivation. `hkdf` takes two arguments: the first is the source of entropy and the second any tag or salt. The `suffix` function is used to derive an initial chain key. Additionally, we use the unary function `h` to model the pre-key hash, which is used during session establishment, see Section 3.

The functions `pqpk`, `encap`, and `decap` model KEM encapsulation and follow the standard symbolic model for asymmetric encryption. Finally, we use the wrapper function `Just` and the constant `None` to model optional values. The function `default` (together with the accompanying equations) allows unpacking an optional value or replacing it with a default, and `unjust` allows the adversary to access the contents of any `Just` value they intercept.

4.3 Properties Specified

4.3.1 Secrecy

PQ3 aims to establish three secrecy properties: message secrecy, forward secrecy, and post-compromise security. In this section, we will unify all these notions in a single property statement. This is formulated as a formula, called a *lemma* in Tamarin.

Security properties in Tamarin are often phrased as: "If some context applies, then either good things happen or the threat model does not apply." For example, secrecy would usually be stated as: "If a participant sends

```
All id sentI me them msg myECDHPk peerECDHPk encapPk #t.
    ( MessageSent(id, sentI, me, them, msg) @ #t
    & SessionSecrets(myECDHPk, peerECDHPk, encapPk) @ #t)
==>   (not Ex #x. K(msg) @ #x)
    | (Ex #x. CompromisedIdentityKey(them) @ #x & #x < #t)
    | ( ( (Ex #x. PQAttack() @ #x)
        | (Ex #x. CompromisedECDHPreKey(them, peerECDHPk) @ #x)
        | (Ex #x. CompromisedECDHKey(id, me, myECDHPk) @ #x)
        | (Ex id2 #x.
             CompromisedECDHKey(id2, them, peerECDHPk) @ #x))
      & ( (Ex #x. CompromisedKemKey(me, encapPk) @ #x)
        | (Ex #x. CompromisedKemKey(them, encapPk) @ #x)
        | (Ex #x. CompromisedKemPreKey(me, encapPk) @ #x)
        | (Ex #x. CompromisedKemPreKey(them, encapPk) @ #x)))
```

Figure 6: Secrecy lemma.

a message, then either that message remains secret or the threat model does not apply." Typically, the threat model not applying means that the adversary has done something that the specification does not aim to protect against. Following this pattern, it might appear natural to phrase forward secrecy and post-compromise security in the same way. However, we find it much clearer to treat forward secrecy and post-compromise security as parts of the property's threat model instead of factoring them out.

Figure 6 contains our secrecy lemma. It states that the adversary must have either compromised the recipients' signing key in the past or compromised one of the most recently sampled ECDH secrets and the most recently sampled encapsulation key in order to learn a message. Both the most recent ECDH secrets and the most recent encapsulation keys are referenced in `SessionSecrets`. Note that the compromise of the most recently sampled ECDH secret may occur due to direct compromise or due to a quantum computing attack from a harvest now, decrypt later adversary. Proving this statement establishes forward secrecy as we show that to learn past messages, the adversary must know past keys. It also establishes post-compromise security because to learn future messages, the adversary must compromise future keys. In both cases, we show that the compromise of current secrets is not enough to learn future or past messages.

Concretely, our lemma entails while the adversary does not possess a quantum computer:

- **Forward secrecy w.r.t. either peer's long-term identity keys.** All messages exchanged before the compromise of the long-term private key associated with the public identity key (which is distributed via secure IDS) of either party stay secret from the adversary.

- **Forward secrecy and post-compromise security w.r.t. ECDH ephemeral keys.** For the compromise of either parties' ECDH ephemeral private keys, we have both forward secrecy, i.e., older messages are secret, and post-compromise security, i.e., as soon as a new ECDH ephemeral key is used, future messages are again secret.

- **Post-compromise security for ECDH pre-keys.** Just like other ECDH keys, if the initial receiver's ECDH private pre-key was compromised, messages that are exchanged after a new ECDH ephemeral key is generated by the recipient, are secret.

Moreover, should the adversary at some point become able to break all non-Kyber keys using quantum computers, PQ3 still provides:

- **Post-quantum forward secrecy and post-compromise security w.r.t. Kyber keys.** Even if a Kyber private key is broken, all messages before this Kyber private keys' usage and all messages after another Kyber private key was used (for encapsulation of a KEM shared secret) remain secret.

In summary, the proof in Tamarin of our secrecy lemma establishes that in the absence of the sender or recipient being compromised, all keys and messages transmitted are secret. Moreover the secrecy property is fine-grained in that compromises can be tolerated in a well-defined sense where the effect of the compromise on the secrecy of data is limited in time and effect as detailed above.

Additional Secrecy Properties. The previous lemma gives us very strong secrecy guarantees. We conjecture though that PQ3 provides even stronger secrecy properties than those established by our secrecy lemma. Due to limitations of Tamarin, we do not have machine checked proofs yet; we discuss this further in Section 4.4.

- **Forward secrecy for ECDH pre-keys.** PQ3 requires that participants update their pre-keys registered at the IDS every 2 weeks. As soon as a client registers a new pre-key, they establish forward secrecy for all previous session initialization messages sent to them.

```
All id i s r m #t.
    MessageReceived(id, i, s, r, m) @ #t
==>   (Ex id_ #x. MessageSent(id_, i, s, r, m) @ #x & #x < #t)
    | (Ex #x. CompromisedIdentityKey(s) @ #x & #x < #t)
```

Figure 7: Agreement lemma.

- **Forward secrecy w.r.t. chain keys.** The compromise of a particular such chain key does not impact the secrecy of any messages exchanged prior to the compromise. Furthermore, see next point.
- **Post-compromise security w.r.t. chain keys.** Even if a particular chain key has been compromised, after the next asymmetric ratchet with the expected partner, future messages are again secret (assuming no further explicit compromise). Only messages sent on this chain before the next ratchet happens will be available to the adversary.
- **Forward secrecy and post-compromise security w.r.t. message keys.** Similarly to chain keys, the compromise of a message key does only impact the secrecy of the message that was encrypted with that message key as every message key is derived from a distinct chain key.

Note that forward secrecy and post-compromise security with respect to the chain and message keys also holds after the adversary has gained access to a quantum computer since the chain and message keys depend on KEM-encapsulated secrets.

4.3.2 Agreement

In contrast to secrecy, formalizing agreement is straight-forward and adapts our definition from Section 2.1 directly. Our formalization of agreement as a Tamarin lemma is provided in Figure 7. The lemma, proven in Tamarin, states that whenever a participant `r` receives a message `m`, apparently from `s` and with message counter `i`, then either `s` had previously sent `m` to `r` with counter `i` or that senders' long-term identity has been compromised in the past.

Additional Agreement Properties. Recall that PQ3 also aims to provide replay protection. Replay protection is usually formalized as *injective agreement*, where "injective" means that there is a one-to-one mapping from receive-events to send-events. We formalized injective agreement as shown in

```
All id1 id2 i1 i2 s1 s2 r1 r2 m #t1 #t2.
    ( MessageReceived(id1, i1, s1, r1, m) @ #t1
    & MessageReceived(id2, i2, s2, r2, m) @ #t2)
==> ( ( (Ex id3 #x. MessageSent(id3, i1, s1, r1, m) @ #x)
      | (Ex #x. CompromisedIdentityKey(s1) @ #x))
    & ( (#t1 = #t2)
      | (Ex #x. CompromisedIdentityKey(s2) @ #x)))
```

Figure 8: Injective agreement lemma.

Figure 8. Note that the formalization is close to our agreement property from earlier, but we quantify over two receive events in the implication's premise. Next to stating that there must be a corresponding send-event for one of the receive-events (as in agreement), the lemma requires that any two receive events of the same message must be the same, unless the long-term identity key of the second receive-event's sender was compromised. Compromise of one sender suffices to violate injective agreement because agreement does not entail secrecy. The adversary could learn a message by compromising the ECDH and KEM keys of the session. They could then send the message again, which requires compromise of a long-term identity key, however, to produce the necessary signature.

We were not able to prove injective agreement in Tamarin for the same reasons as for the additional secrecy properties just discussed, and we will explain these limitations in Section 4.4. Nevertheless, our analysis supports that PQ3 provides injective agreement as well. This is because:

(a) For every counter and for a fixed session, participants accept a message only once since they immediately increase the counter. By our agreement lemma, we know participants agree on message counters, and therefore, the adversary cannot simply replay an honestly generated and encrypted message under a different counter in the same session.

(b) Moreover, honest participants will never reuse keys to send a message as every session is initialized with one freshly sampled ECDH ephemeral key, and therefore, the adversary also cannot replay the ciphertext in a different session.

(c) The message key is included in the signature, i.e., the adversary also cannot generate their own ciphertext and inject that as this would require compromise of the sender's long-term identity key, which is not considered under our threat model as this key is stored in the Apple Secure Enclave.

We plan to investigate and prove these statements formally in future work.

4.4 Limitations

The scope of our analysis was limited to the protocol design as described in the documentations we received. PQ3's implementation was not part of this analysis. Furthermore, as our analysis is based on symbolic models, it necessarily abstract away some details of the concrete implementation, like message lengths and the particular algorithmic details of the ciphers used.

Due to technical limitations, we do not explicitly consider the compromise of root keys, chain keys, and message keys. We instead allow the compromise of the secret inputs used to derive these keys. This limits our reasoning about an adversary with crypt-analytic capabilities that can break such keys, and what we can derive about remaining guarantees by, e.g., symmetric ratcheting. This limitation stems from what we believe to be limitation of Tamarin. For example, when we considered the explicit leakage of, e.g., chain keys, Tamarin failed to deduce the temporal dependencies between various session states, i.e., it would not conclude that it is impossible to reuse a counter in a session using the exact same key material, although the counter is incremented on every interaction. This limitation also prevented us from proving injective agreement. We believe it should be possible to soundly extend Tamarin to better support such reasoning. However this remains as future work.

Lastly, we omit multiple incoming chains from the model, do not consider long-term identity or pre-key rollover, and only consider group chats implicitly. However, PQ3 implements group chats using multiple, individual device-to-device sessions, and our analysis considers the security of each of these sessions.

5 Results and Conclusions

We have used Tamarin to formally verify the device-to-device messaging protocol PQ3. From our analysis, we conclude that this protocol achieves strong security guarantees against an active network adversary who can selectively compromise parties and has quantum computing capabilities. Our formal analysis provides machine-checked proofs of fine-grained secrecy and authentication properties, and we found no attacks on PQ3 during the development of our proofs. Our analysis provides a high degree of assurance that

PQ3 functions securely, even when sufficiently powerful quantum computers become available.

Our formal analysis provides value beyond the proofs themselves. Our verified security guarantees are far more detailed than what was set out as security goals in PQ3's specification. For example, our analysis establishes that PQ3 provides post-compromise security even against active adversaries provided that long-term identity keys do not leak. In contrast, Signal, for example, only provides post-compromise security against passive adversaries. Additionally, the secrecy lemma provides a detailed account of the impact that the compromise of every individual key has. Finally, as is often the case when using Formal Methods, our analysis helped tighten PQ3's specification. During the specification's formalization, we uncovered both ambiguities and technical issues, which have since been addressed by Apple. These include, for example, ambiguities around key usage or checks made by participants that could not have been implemented as originally specified.

Next Steps and Future Work. We plan to make our formal models and proofs openly available. We also intend to address the shortcomings of our formal model in an academic publication.

References

[1] Apple Security Engineering and Architecture (SEAR). Advancing iMessage security: iMessage Contact Key Verification - Apple Security Research.

[2] Karthikeyan Bhargavan, Abhishek Bichhawat, Quoc Huy Do, Pedram Hosseyni, Ralf Küsters, Guido Schmitz, and Tim Würtele. DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code. pages 523–542, 2021.

[3] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[4] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. Revision 1, 2016-11-20.

[5] Benedikt Schmidt, Simon Meier, Cas Cremers, and David A. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94. IEEE Computer Society, 2012.