A quantum-safe circuit-extension handshake for Tor

John Schanck[1,2], William Whyte[1], and Zhenfei Zhang[1]

[1] Security Innovation, Wilmington, MA 01887
[2] University of Waterloo, Waterloo, Canada
{jschanck,wwhyte,zzhang}@securityinnovation.com

Abstract. We propose a method for integrating NTRUEncrypt into the ntor key exchange protocol as a means of achieving a quantum-safe variant of forward secrecy. The proposal is a minimal change to ntor, essentially consisting of an NTRUEncrypt-based key exchange performed in parallel with the ntor handshake. Performance figures are provided demonstrating that the client bears most of the additional overhead, and that the added load on the router side is acceptable.
We make this proposal for two reasons. First, we believe it to be an interesting case study into the practicality of quantum-safe cryptography and into the difficulties one might encounter when transitioning to quantum-safe primitives within real-world protocols and code-bases. Second, we believe that Tor is a strong candidate for an early transition to quantum-safe primitives; users of Tor may be justifiably concerned about adversaries who record traffic in the present and store it for decryption when technology or cryptanalytic techniques improve in the future.

1 Introduction

A key exchange protocol allows two parties who share no common secrets to agree on a common key over a public channel. In addition to achieving this basic goal, key exchange protocols may satisfy various secondary properties that are deemed important to security in particular settings. Modern key exchange protocols typically satisfy some of the following properties:

One-way or mutual authentication. A protocol achieves mutual authentication if both parties executing it can be assured of their peer's identity. Protocols such as [4], [13], and [14] must assume that each party possesses a certified copy of their peer's public key in order to achieve this goal. While desirable, mutual authentication is often difficult to achieve in practice, and the weaker property of one-way authentication, in which only one party is authenticated, is more common.

Anonymity. One-way authentication is well suited for networks, such as Tor [6], that aim to provide their clients with strong anonymity guarantees. In such systems, one party (usually the server) publishes a long-term identity key that may be used for authentication, while the other party (the client) remains anonymous. One-way anonymity is provided by some Key Encapsulation Mechanisms (KEMs) such as [12,23], as well as some Diffie-Hellman protocols, such as ntor [8].

Forward secrecy. A protocol achieves forward secrecy if the compromise of any party's long-term key material does not affect the secrecy of session keys derived prior to said compromise. This property is typically achieved by mixing long-term key material with ephemeral, single-use, keys. It is an essential requirement for some applications, particularly those where an attacker may be able to store encrypted data for long periods of time until legal, technological, or cryptanalytic means become available for revealing keys.

This feature is more and more desirable with the advent of quantum computers, through which cryptanalytic compromise of long-term keys may become a real possibility rather than mostly theoretical concern. There are currently no widely deployed key-exchange mechanisms capable of resisting quantum adversaries.

(Forward) quantum-resistance. A protocol is quantum-resistant (or quantum-safe) if it remains secure under the assumption that the adversary can perform polynomial time quantum computations. There are no widely deployed quantum-safe key exchange protocols in use today. All methods based

on discrete log (Diffie-Hellman, ECDH) and integer factorization (RSA) can be broken in polynomial time using quantum Fourier sampling techniques [19,20].

There are several proposals for quantum-safe key exchange mechanisms in the literature, including several direct constructions of Diffie-Hellman-like protocols from problems thought to be hard for quantum computers [10,16,3]. Another approach, the one taken here, is to instantiate a key-encapsulation mechanism with a quantum-safe encryption primitive such as NTRUEncrypt [9,25]. An example of such an instantiation was proposed in [23].

In order for these schemes to be fully quantum-resistant they would need to maintain their secondary attributes in the presence of quantum adversaries. For instance, authentication could be achieved using a pre-shared symmetric key or a quantum-safe signature scheme, however both approaches present practical challenges. In the short term it seems reasonable to investigate key exchange mechanisms that do not provide quantum-safe authenticity, but that otherwise resist active classical adversaries and passive quantum adversaries. We will call such schemes *forward quantum-resistant*. The scheme presented in [3] and the one presented here both achieve this property.

Disaster-resistance. We say that a protocol is disaster-resistant if its security rests on a heterogeneous set of assumptions in such a manner that the failure of any one assumption would not compromise the security of the entire scheme. This is an especially desirable property when deploying new cryptographic primitives.

1.1 Our contribution

We demonstrate how to incorporate NTRUEncrypt into the ntor protocol as a means of achieving forward quantum-resistance. The resulting scheme is easily seen to inherit the forward secrecy and one-way anonymity properties of ntor.

We propose an instantiation of our scheme at the 128-bit security level that uses ntruees439ep1 in addition to the primitives present in the production instantiation of ntor. We have implemented our proposal within the existing Tor codebase, and have made our implementation freely available [18].

The primary disadvantage of our scheme is the increased byte-size of the handshake messages; NTRUEncrypt keys and ciphertexts at the recommended security level are approximately 600 bytes. Unfortunately this exceeds the 512-byte cell size for the Tor protocol, so incorporating our handshake into Tor would not be entirely trivial and would require either the definition of a new control message, or an increase in cell size.

Furthermore, since we have avoided heavy cryptographic methods such as quantum-resistant signatures, our protocol does not provide security against active quantum adversaries. Fully quantum-resistant key exchange may be required in some settings, but we believe that a security model that includes passive, but not active, quantum adversaries is realistic for the near future.

Paper Organization In the next section, we review the background necessary for this paper. In Section 3, we review the building blocks of our protocol. The protocol will be presented in Section 4 and its security will be analyzed in Section 5. In Section 6, we compare the performance of our protocol with ntor, and in Section 7 we explore the feasibility of integrating our handshake into the production Tor environment.

2 Background

2.1 Notation

In the rest of the paper, $\mathbb{G}$ is always a cyclic group of known prime order q, and g is a fixed generator of $\mathbb{G}$. We use multiplicative notation for group operations. Sampling the uniform distribution on a set $\mathbb{X}$ is denoted by $x \leftarrow_R \mathbb{X}$. We freely associate any object with a bitstring representing it, for instance $\mathsf{Hash}(g^x)$ is presumed to be well defined and unambiguous. The concatenation of the strings a and b is denoted by $a|b$.

The protocols we will discuss involve two honest parties who we will call Alice and Bob. Their identities are represented by $\hat{A}$ and $\hat{B}$. In a client-server scenario, Bob is the server and Alice is the client. Party $\hat{P}$ has access to a memory $M_{\hat{P}}$ in which they can store session state. The state for session Ψ is denoted $M_{\hat{P}}[\Psi]$.

2.2 Cryptographic primitives

Public key primitives The protocols described below involve both Diffie-Hellman and NTRUEncrypt operations and thus make use of the following PPT algorithms. Relevant parameters, $\mathbb{G}, q, g$ for Diffie-Hellman and $\mathbb{M}$ for NTRUEncrypt, are implicitly defined as functions of the security parameter λ.

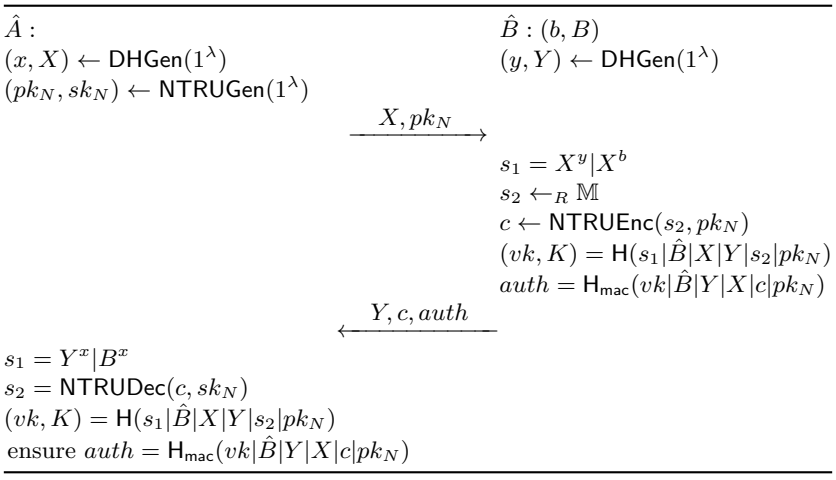

Fig. 1. The proposed protocol: ntrutor

- $\mathsf{DHGen}(1^\lambda)$: Let $x \leftarrow_R [1, q-1]$, and $X = g^x$. Outputs the Diffie-Hellman keypair (x, X), where x is the private key and X is the public key.
- $\mathsf{NTRUGen}(1^\lambda)$: Outputs an NTRUEncrypt keypair (sk, pk) where sk is the secret key and pk is the public key.
- $\mathsf{NTRUEnc}(m, pk)$: Takes as input a message $m \in \mathbb{M}$, and an NTRUEncrypt public key pk. Outputs a ciphertext c.
- $\mathsf{NTRUDec}(c, sk)$: Takes as input a ciphertext c, and an NTRUEncrypt secret key sk. Outputs a message $m \in \mathbb{M}$.

Key derivation functions A Key Derivation Function (KDF) [22,1] is a function that takes three inputs and outputs a string of ℓ bits. The three inputs are: a sample from a source of keying material, $K \in \mathcal{K}$; a sample from a set of possible salt values, $S \in \mathcal{S}$; and a bitstring specifying additional, or contextual, information, I. It is understood that the source from which the keying material is derived leaks some information to the environment[1], so the role of a key derivation function is to ensure that, despite this inevitable leakage, the ℓ output bits are uniformly random.

Krawczyk presented an instantiation of a KDF based on a Hash-based Message Authentication Code (HMAC) in [11] and provided a formal definition of security for KDFs called *m-entropy security*. This definition captures the idea that the output of a KDF should be indistinguishable from a uniform ℓ bit string so long as that the conditional min-entropy of the keying material, given the naturally leaked information, is at least m bits.

The KDF appearing in our protocol is assumed to be λ-entropy secure.

2.3 Related work

From Diffie-Hellman to ntor. Two parties, Alice and Bob, who have publicly agreed on parameters – namely a generator g of a group $\mathbb{G}$ of prime order q – may derive a shared secret in the presence of passive eavesdroppers using the Diffie-Hellman protocol [5].

Alice selects x in $[1, q-1]$ and sends $X = g^x$ to Bob. Similarly, Bob selects y in $[1, q-1]$ and sends $Y = g^y$ to Alice. They arrive at the common value g^{xy} by computing Y^x and X^y respectively.

The security of this protocol requires that the decisional Diffie-Hellman assumption holds for the group $\mathbb{G}$. That is, given $g, g^x, g^y \in \mathbb{G}$, the element g^{xy} is indistinguishable from an element chosen uniformly at random from $\mathbb{G}$. This is one of the core assumptions of modern cryptography; its apparent validity with respect to non-quantum distinguishers for some cyclic groups has enabled many cryptographic schemes.

The authenticated version of the Diffie-Hellman protocol presented in Figure 2 was formally analyzed by Shoup in [21], although it was likely known prior

[1] For instance, a Diffie-Hellman handshake might use g^{xy} as keying material and leak g^x, g^y and the group parameters to the environment.

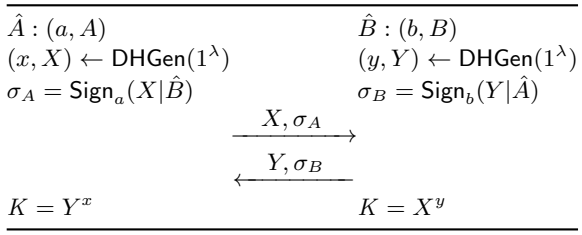

Fig. 2. The signed Diffie-Hellman key exchange protocol.

to that analysis. It is sometimes referred to as the signed Diffie-Hellman protocol.

In this protocol each party must produce a signature on their public group element and their peer's identity. By verifying Alice's signature, Bob is convinced that the group element he received has come from Alice, and vice versa.

Signed Diffie-Hellman suffers from several shortcomings, the most troubling being that leakage of an ephemeral key allows an adversary to impersonate the leaked key's owner in subsequent sessions.

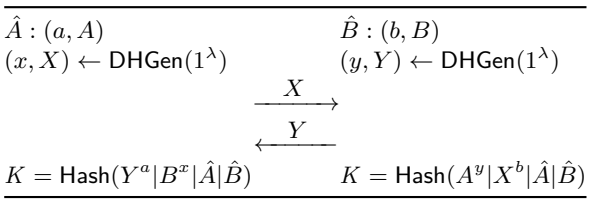

Fig. 3. The KEA+ key exchange protocols

This and other weaknesses are addressed in the KEA+ protocol of Lauter and Mityagin [14]. KEA+ avoids the aforementioned impersonation attack by deriving the shared secret from a combination of long- and short-term key material contributed by both parties (see Figure 3). Specifically, the parties derive two shared secrets g^{ay}, and g^{bx} where a, b are long-term secrets and x, y are short-term secrets. These values are hashed, along with the identities of both parties, produce the final key. The inclusion of the identities is crucial for preventing unknown key share attacks[14].

Finally, we have arrived at ntor [8], the one-way authenticated key exchange protocol that is used in recent versions of Tor [6]. The ntor protocol can be seen as a variant of KEA+ in which Alice does not reveal a long-term secret, is not authenticated, and is allowed to remain anonymous. As detailed in Figure 4, the parties derive two shared secrets, the first g^{xy} combines the parties' short-term key material, and the second g^{bx} mixes Alice's short-term key with Bob's long-term key. The latter value ensures that Alice maintains the ability to authenticate Bob, and the former provides forward secrecy against leakage of Bob's long-term key.

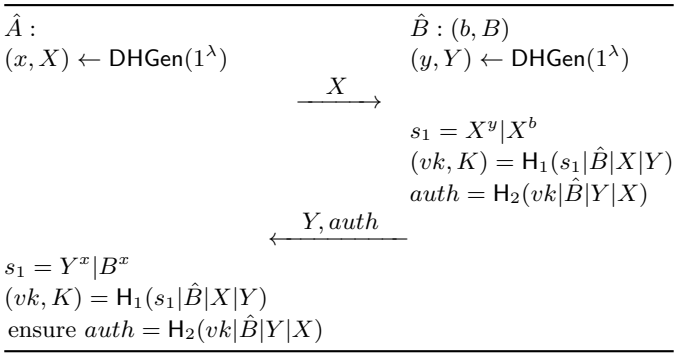

Fig. 4. The ntor protocol

Key encapsulation mechanisms. Diffie-Hellman protocols are far from the only method by which two parties may derive a common key over a public channel. Among the many alternatives are Key Encapsulation Mechanisms (KEMs).

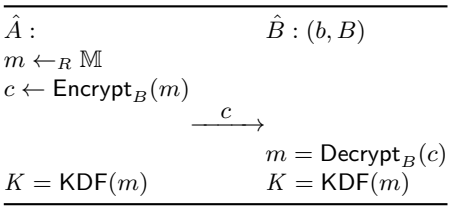

Fig. 5. The key encapsulation mechanisms

In a KEM, Alice encrypts a random message to Bob using Bob's long-term public key. Bob then decrypts the received ciphertext and the parties derive a shared secret from Alice's message using a Key Derivation Function (KDF). Such a KEM provides one-way authentication and one-way anonymity: Alice may remain anonymous during the execution of

the protocol, as the shared secret does not depend on any value linked to her identity; and Alice is able to authenticate Bob, as she has an authentic copy of his public key and only he can decrypt her message. Forward secrecy, however, is notably lacking. If Bob's long-term key is compromised then confidentiality is lost for every session previously established.

3 Security model

The ntor protocol was analyzed in a variant of the extended Canetti-Krawczyk (eCK) model with support for one-way authentication [8]. For continuity with this work we will use essentially the same model, but we must make a slight modification in order to argue for quantum-safe forward-secrecy. Fortunately most of the machinery needed for this was developed in [15], which proposed an extension to the model of [8] for the purpose of analyzing authenticated quantum key expansion protocols. In the model of [15] (hereafter MSU) all parties, including the adversary, have access to a quantum Turing machine capable of executing algorithms with runtime bounded by $t_q(\lambda)$ and memory bounded by $m_q(\lambda)$. The inclusion of explicit bounds allows us to more accurately model the types of quantum computations which are feasible today and in the near future.

3.1 Communication and adversary model

In the MSU model a *party* is an interactive classical Turing machine. A party has access to a memory, a random tape, and a bounded time/memory quantum Turing machine.

Each party has a public identifier represented by a capital letter with a hat (e.g. $\hat{A}$). These identifiers are used for routing communication and for registering certificates. There is a privileged party, labeled by $\hat{C}$, that serves the role of certificate authority and with whom all communication is guaranteed to be authentic. As in the eCK model, parties do not prove knowledge of a private key when registering a public key with the certificate authority. This allows the adversary to bind an arbitrary public key to an identity they control, even if that public key is owned by another party. The adversary cannot, however, register a public key for a party that they do not control.

Parties may have several asymmetric value pairs in memory at any given time. These are denoted by (x, X) where x is a secret value, such as a private key, and X is a public label for x (often the corresponding public key).

Parties communicate with each other via *activation requests*. These requests are created either directly by the adversary or in response to previous requests from the adversary. The parties are assumed to execute the protocol honestly, but the adversary can record, modify, delete, or attempt to forge requests made by other parties. A *session* is started when the adversary requests that a party initiate a protocol with another party of the adversary's choosing. Each party participating in a session ascribes a locally unique session identifier, $\Psi_{\hat{P}}$, to the session. Session identifiers are known to the adversary.

After running their respective parts of the protocol, the participating parties output either an error symbol, $\perp$, or a tuple of the form $(sk, pid, \boldsymbol{v}, \boldsymbol{u})$. Once all parties have produced an output the session is considered *completed*. Prior to completion a session is called *active*.

The values in a party's output tuple,[2] $(sk, pid, \boldsymbol{v}, \boldsymbol{u})$, respectively identify: the session key, the identity of the peer with whom the session key is believed to have been established, a nested list of the public values used to derive sk, and a nested list of the public values used for authenticating the peer pid. If the peer was unauthenticated (anonymous) during the execution of the protocol then the token $\circledast$ is used for pid and the list $\boldsymbol{u}$ will be empty.

The following activation requests are defined:[3]

- Req$(id, command, arguments, protocol)$: This activation request directs the intended recipient (specified by id) to perform the action specified by $command$ using $arguments$ as input. The $protocol$ is included to ensure that the command is well defined. This is the only request type that

[2] The output tuple was introduced in [8] as an enhancement to traditional AKE security models where the adversary-learnable values must be specified at the model level. The output tuple encodes which values are learnable at the protocol level, and thereby allows for the comparison of protocols that would have been incomparable in earlier AKE models.

[3] Several more requests are defined in the full MSU model; for simplicity we have omitted requests for describing quantum communication and the requests describing the interaction of classical and quantum Turing machines. We have also merged the two variants of the SendC request from MSU into our Req request, and added id parameters where they were implicit in MSU. These are purely syntactic changes.

ordinary parties can issue, the rest may only be issued by the adversary.

- $\mathsf{RevealNext}(id, type) \rightarrow X$: This request allows the adversary to learn the *public* value of the specified *type* that the party *id* will use next. For instance, $\mathsf{RevealNext}(\hat{A}, \mathsf{DH})$ causes $\hat{A}$ to generate a new Diffie-Hellman key $(x, X) \leftarrow \mathsf{DHGen}$ and to return X to the adversary. The pair (x, X) is marked as unused and the next time $\hat{A}$ would call DHGen (in response to a request other than $\mathsf{RevealNext}$) it will retrieve (x, X) instead. Successive $\mathsf{RevealNext}$ queries allow the adversary to learn the next k public values of any type that the party will use.
- $\mathsf{Partner}(id, label) \rightarrow x$: This request allows the adversary to learn the secret value associated with the given public label. For instance, in response to $\mathsf{Partner}(\hat{A}, X)$ the Turing machine $\hat{A}$ returns x. The session key is labeled by the session ID its owner ascribes to it, i.e. the adversary can learn the key for a session Ψ owned by the party $\hat{A}$ by querying $\mathsf{Partner}(\hat{A}, \Psi)$.

The adversary can issue any number of these requests in any order.

Partnering to a value is a very important concept in this model.

Definition 1 (Partnering). *If (x, X) is a value pair owned by $\hat{A}$, then the adversary is said to be a partner to X if and only if it has queried* $\mathsf{Partner}(\hat{A}, X)$.

The structure of honest parties' output vectors, i.e. the segregation of labeled values into those associated with keying material, $\boldsymbol{v}$, and those associated with authentication, $\boldsymbol{u}$, allows for fine grained control over which values the adversary may learn through partnering as well as when the adversary may them. With the exception of the session key, labeled values that appear in neither $\boldsymbol{v}$ nor $\boldsymbol{u}$ are not able to be learned by the adversary through partnering.

3.2 Security definitions

We now give the security definitions that will be used in our security arguments in Section 5.

Definition 2 (Correctness [15]). *A key exchange protocol is said to be* correct *if, when all protocol messages are relayed faithfully, without changes to content or ordering, the peer parties output the same session key K and vector $\boldsymbol{v}$.*

Security will be defined with respect to a game the adversary plays after making some (polynomial in λ) number of activation requests and observing/manipulating the honest parties' results. The adversary starts the game by issuing the following query to an oracle:

- $\mathsf{Test}(id, \Psi) \rightarrow \{0,1\}^\lambda$: If the party specified by *id* has not output a vector for session Ψ the oracle returns $\perp$. Otherwise, the oracle chooses $b \leftarrow \{0,1\}$ uniformly. If $b = 1$ it returns the session key corresponding to Ψ. If $b = 0$ it returns a uniform random string in $\{0,1\}^\lambda$.

The adversary may only issue one Test query.

Definition 3 (Fresh session [15]). *A session Ψ owned by an honest party $\hat{P}_i$ is* fresh *if all of the following occur:*

1. *For every vector $\boldsymbol{v}_j$, in $\hat{P}_i$'s output for session Ψ, there is at least one element X in $\boldsymbol{v}_j$ such that the adversary is not a partner to X.*
2. *The adversary did not issue $\mathsf{Partner}(\hat{P}_j, \Psi')$ to any honest party $\hat{P}_j$ for which Ψ' has the same public output vector as Ψ (including the case where $\Psi' = \Psi$ and $\hat{P}_j = \hat{P}_i$.*
3. At the time of session completion*, for every vector $\boldsymbol{u}_j$, in $\hat{P}_i$'s output for session Ψ, there was at least one element X in $\boldsymbol{u}_j$, such that the adversary was not a partner to X.*

Note that the session is not fresh if either $\boldsymbol{v}$ or $\boldsymbol{u}$ is empty. In particular sessions established with anonymous peers are not fresh.

Definition 4 (Security [15]). *Let λ be a security parameter. An authenticated key exchange protocol is* secure *(or $(t_c(\lambda), t_q(\lambda), m_q(\lambda))$-secure) if, for all adversaries $\mathcal{A}$ with classical runtime bounded by $t_c(\lambda)$, quantum runtime bounded by $t_q(\lambda)$, and quantum memory bounded by $m_q(\lambda)$, the advantage of $\mathcal{A}$ in guessing the bit b used in the Test query of a fresh session is negligible in the security parameter; in other words, the probability that $\mathcal{A}$ can distinguish the session key of a fresh session from a random string of the same length is negligible in λ.*

Freshness delineates the situations in which security is relevant. Note that with these definitions of freshness and security the adversary can partner to some of the keying material from each $\boldsymbol{v}_i$, and preserve the freshness of the session, either while the session is active or after the session is complete, but

cannot partner to all values in any $\boldsymbol{v}_i$ at any time. The adversary is similarly limited in the $\boldsymbol{u}_i$ components to which it can be partnered while a session is active, but is allowed to partner to the entire $\boldsymbol{u}$ vector after completion.

Definition 5 (Forward-secrecy). *An authenticated key exchange protocol provides* forward secrecy *if it is secure under Definition 4 and for every fresh session Ψ the following conditions are met:*

1. *Every long-term value used by an honest party during the execution of session Ψ is labeled by at least one component of $\boldsymbol{u}$.*
2. *If the adversary is not partnered to any component of $\boldsymbol{v}$, then Ψ would remain fresh if the adversary partnered to every component of $\boldsymbol{u}$.*

Definition 6 (Quantum-resistance). *An authenticated key exchange protocol provides* quantum-resistance *if it is $(t_c(\lambda), t_q(\lambda), m_q(\lambda))$-secure for polynomially bounded $t_c(\lambda) = t_q(\lambda) = m_q(\lambda)$.*

In analogy with the definition of *long-term security* provided in [15] we propose the following definition of *forward quantum-resistance*. This definition aims to capture the possibility of an adversary who, in an attempt to win the Test game, passes a transcript of observed activation requests to a collaborator that has access to a more powerful quantum Turing machine.

Definition 7 (Forward quantum-resistance). *Let π be a $(t_c(\lambda), t_q(\lambda), m_q(\lambda))$-secure authenticated key exchange protocol. Let $\mathcal{A}$ be an adversary as in Definition 4, let $\kappa \in \{0,1\}^\lambda$ be the result of $\mathcal{A}$'s query, $\mathsf{Test}(\hat{P}, \Psi)$, on a fresh session Ψ. Finally let T be a transcript of classical and quantum bits output by $\mathcal{A}$ after a $(t_c(\lambda), t_q(\lambda), m_q(\lambda))$-bounded computation.*

We say π is forward quantum-resistant with respect to $\mathcal{A}$ *if, for all quantum Turing machines $\mathcal{M}$ with runtime bounded by $t'_q = t_c(\lambda)$ and memory bounded by $m'_q = t_c(\lambda)$, the advantage of $\mathcal{M}$, given (T, κ), in guessing the bit b that was used in the Test query is negligible in λ.*

We say that π is forward quantum-resistant *if it is forward quantum resistant with respect to all adversaries $\mathcal{A}$ meeting the above criteria.*

4 Protocols

4.1 The ntor protocol

The general outline of ntor was provided in the Section 2.3. So as to fully illustrate our method, we first present the construction from [8] using the model of [15] before presenting our protocol.

The protocol identifier ntor implicitly defines a security parameter, λ, Diffie-Hellman parameters, and two hash functions:

$$\mathsf{H}_{\mathsf{mac}} : \{0,1\}^* \to \{0,1\}^\lambda$$
$$\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda \times \{0,1\}^\lambda$$

Under normal operation the ntor protocol can be modeled by the following sequence of activation requests involving the parties $\hat{A}$, $\hat{B}$ and the certificate authority $\hat{C}$:

1. $\mathsf{Req}(\hat{B}, \text{“}init\ server\text{”}, (\varnothing), \mathsf{ntor})$,
2. $\mathsf{Req}(\hat{A}, \text{“}fetch\ certificates\text{”}, (\varnothing), \mathsf{ntor})$,
3. $\mathsf{Req}(\hat{A}, \text{“}start\text{”}, (\hat{B}), \mathsf{ntor})$,
4. $\mathsf{Req}(\hat{B}, \text{“}respond\text{”}, (\Psi_{\hat{A}}, \hat{A}, \hat{B}, X), \mathsf{ntor}))$,
5. $\mathsf{Req}(\hat{A}, \text{“}finish\text{”}, (\Psi_{\hat{A}}, Y, auth), \mathsf{ntor})$.

At each step the parties' behavior is governed by the following rules:

1. On $\mathsf{Req}(\hat{B}, \text{“}init\ server\text{”}, (\varnothing), \mathsf{ntor})$ $\hat{B}$:
   - Generates a long-term keypair,
     $$(b, B) \leftarrow \mathsf{DHGen}(1^\lambda).$$
   - Issues $\mathsf{Req}(\hat{C}, \text{“}register\text{”}, (B, \hat{B}), \mathsf{ntor})$.

2. On $\mathsf{Req}(\hat{A}, \text{“}fetch\ certificates\text{”}, (\varnothing), \mathsf{ntor})$ $\hat{A}$:
   - Retrieves a list of all registered certificates from $\hat{C}$.
   - Stores the received certificates in memory,
     $$M_{\hat{A}}[\text{“}certs\text{”}] \leftarrow (cert_1, \ldots, cert_n).$$

3. On $\mathsf{Req}(\hat{A}, \text{“}start\text{”}, (\hat{B}), \mathsf{ntor})$ $\hat{A}$:
   - Searches $M_{\hat{A}}[\text{“}certs\text{”}]$ for a valid certificate for $\hat{B}$ or outputs $\perp$ if none is found.
   - Creates a new session, $\Psi_{\hat{A}}$.
   - Generates an ephemeral DH keypair
     $$(x, X) \leftarrow \mathsf{DHGen}(1^\lambda).$$
   - Stores $M_{\hat{A}}[\Psi_{\hat{A}}] \leftarrow (\hat{B}, (x, X), \mathsf{ntor})$.
   - Issues $\mathsf{Req}(\hat{B}, \text{“}respond\text{”}, (\Psi_{\hat{A}}, \hat{A}, \hat{B}, X), \mathsf{ntor}))$.

4. On $\mathsf{Req}(\hat{B}, \text{“}respond\text{”}, (\Psi_{\hat{A}}, \hat{A}, \hat{B}, X), \mathsf{ntor}))$ $\hat{B}$:
   - Verifies $X \in \mathbb{G}$, or outputs $\perp$.
   - Creates a new session, $\Psi_{\hat{B}}$.

- Generates an ephemeral DH keypair
$$(y, Y) \leftarrow \mathsf{DHGen}(1^\lambda).$$
- Sets $s_1 = X^y | X^b$.
- Sets $(vk, K) = \mathsf{H}(s_1|B|X|Y|\mathsf{ntor})$.
- Sets $auth = \mathsf{H}_{\mathsf{mac}}(vk|B|Y|X|\mathsf{ntor}|\text{“Server”})$.
- Issues $\mathsf{Req}(\hat{A}, \text{“}finish\text{”}, (\Psi_{\hat{A}}, Y, auth), \mathsf{ntor})$.
- Deletes y, s_1.
- Outputs $(K, \circledast, ((X), (Y, B)), ((\varnothing)))$.

5. On $\mathsf{Req}(\hat{A}, \text{“}finish\text{”}, (\Psi_{\hat{A}}, Y, auth), \mathsf{ntor})$ $\hat{A}$:
   - Verifies $M_{\hat{A}}[\Psi_{\hat{A}}]$ exists or outputs $\perp$.
   - Verifies $Y \in \mathbb{G}$ and that c is a valid ciphertext or outputs $\perp$.
   - Sets $s_1 = Y^x | B^x$.
   - Sets $(vk, K) = \mathsf{H}(s_1|B|X|Y|\mathsf{ntor})$.
   - Ensures $auth =$
   $$\mathsf{H}_{\mathsf{mac}}(vk|B|Y|X|\mathsf{ntor}|\text{“Server”})$$
   or outputs $\perp$.
   - Deletes $M_{\hat{A}}[\Psi_{\hat{A}}]$ and s_1.
   - Outputs $(K, \hat{B}, ((X), (Y, B)), ((B)))$.

If either party outputs $\perp$, it is assumed that both parties abort the protocol and delete all temporary state.

4.2 The proposed protocol

The protocol identifier ntrutor implicitly defines a security parameter, λ, a DH group $\mathbb{G}$, and two hash functions:

$$\mathsf{H}_{\mathsf{mac}} : \{0,1\}^* \to \{0,1\}^\lambda$$
$$\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda \times \{0,1\}^\lambda$$

It additionally specifies a λ-bit secure NTRUEncrypt parameter set.

Under normal operation the ntrutor protocol can be modeled by the following sequence of activation requests involving the parties $\hat{A}$, $\hat{B}$ and the certificate authority $\hat{C}$:

1. $\mathsf{Req}(\hat{B}, \text{“}init\ server\text{”}, (\varnothing), \mathsf{ntrutor})$,
2. $\mathsf{Req}(\hat{A}, \text{“}fetch\ certificates\text{”}, (\varnothing), \mathsf{ntrutor})$,
3. $\mathsf{Req}(\hat{A}, \text{“}start\text{”}, (\hat{B}), \mathsf{ntrutor})$,
4. $\mathsf{Req}(\hat{B}, \text{“}respond\text{”}, (\Psi_{\hat{A}}, \hat{A}, \hat{B}, X, pk_N), \mathsf{ntrutor}))$,
5. $\mathsf{Req}(\hat{A}, \text{“}finish\text{”}, (\Psi_{\hat{A}}, Y, c, auth), \mathsf{ntrutor})$.

At each step the parties' behavior is governed by the following rules:

1. On $\mathsf{Req}(\hat{B}, \text{“}init\ server\text{”}, (\varnothing), \mathsf{ntrutor})$ $\hat{B}$:
   - Generates a long-term keypair,
   $$(b, B) \leftarrow \mathsf{DHGen}(1^\lambda)$$
   - Issues $\mathsf{Req}(\hat{C}, \text{“}register\text{”}, (B, \hat{B}), \mathsf{ntrutor})$.

2. On $\mathsf{Req}(\hat{A}, \text{“}fetch\ certificates\text{”}, (\varnothing), \mathsf{ntrutor})$ $\hat{A}$:
   - Retrieves a list of all registered certificates from $\hat{C}$.
   - Stores the received certificates in memory,
   $$M_{\hat{A}}[\text{“}certs\text{”}] \leftarrow (cert_1, \ldots, cert_n).$$

3. On $\mathsf{Req}(\hat{A}, \text{“}start\text{”}, (\hat{B}), \mathsf{ntrutor})$ $\hat{A}$:
   - Searches $M_{\hat{A}}[\text{“}certs\text{”}]$ for a valid certificate for $\hat{B}$ or outputs $\perp$ if none is found.
   - Creates a new session, $\Psi_{\hat{A}}$.
   - Generates an ephemeral DH keypair
   $$(x, X) \leftarrow \mathsf{DHGen}(1^\lambda).$$
   - Generates an ephemeral NTRU keypair
   $$(sk_N, pk_N) \leftarrow \mathsf{NTRUGen}(1^\lambda).$$
   - Stores
   $$M_{\hat{A}}[\Psi_{\hat{A}}] \leftarrow (\hat{B}, (x, X), (sk_N, pk_N), \mathsf{ntrutor}).$$
   - Issues $\mathsf{Req}(\hat{B}, \text{“}respond\text{”}, (\Psi_{\hat{A}}, \hat{A}, \hat{B}, X, pk_N), \mathsf{ntrutor}))$.

4. On $\mathsf{Req}(\hat{B}, \text{“}respond\text{”}, (\Psi_{\hat{A}}, \hat{A}, \hat{B}, X, pk_N), \mathsf{ntrutor}))$ $\hat{B}$:
   - Verifies $X \in \mathbb{G}$ and that pk_N is a valid NTRUEncrypt key, or outputs $\perp$;
   - Creates a new session, $\Psi_{\hat{B}}$.
   - Generates an ephemeral DH keypair
   $$(y, Y) \leftarrow \mathsf{DHGen}(1^\lambda).$$
   - Samples $s_2 \leftarrow_R \{0,1\}^\lambda$.
   - Encrypts s_2 under pk_N:
   $$c \leftarrow \mathsf{NTRUEnc}(s_2, pk_N).$$
   - Sets $s_1 = X^y | X^b$.
   - Sets $(vk, K) = \mathsf{H}(s_1|B|X|Y|s_2|pk_N|\mathsf{ntrutor})$.
   - Sets $auth =$
   $$\mathsf{H}_{\mathsf{mac}}(vk|B|Y|X|c|pk_N|\mathsf{ntrutor}|\text{“Server”}).$$
   - Issues
   $\mathsf{Req}(\hat{A}, \text{“}finish\text{”}, (\Psi_{\hat{A}}, Y, c, auth), \mathsf{ntrutor})$.
   - Deletes y, s_1 and s_2.
   - Outputs $(K, \circledast, ((X, pk_N), (Y, B, pk_N)), ((\varnothing)))$.

5. On $\mathsf{Req}(\hat{A}, \text{“}finish\text{”}, (\Psi_{\hat{A}}, Y, c, auth), \mathsf{ntrutor})$ $\hat{A}$:
   - Verifies $M_{\hat{A}}[\Psi_{\hat{A}}]$ exists or outputs $\perp$.
   - Verifies $Y \in \mathbb{G}$ and that c is a valid ciphertext or outputs $\perp$.
   - Decrypts c using sk_N and sets
   $$s_2 = \mathsf{NTRUDec}(c, sk_N).$$
   - Sets $s_1 = Y^x | B^x$.
   - Sets $(vk, K) = \mathsf{H}(s_1|B|X|Y|s_2|pk_N|\mathsf{ntrutor})$.

- Ensures $auth =$
  $\mathsf{H}_{\mathsf{mac}}(vk|B|Y|X|c|pk_N|\mathsf{ntrutor}|\text{“Server”})$
  or outputs $\perp$.
- Deletes $M_{\hat{A}}[\Psi_{\hat{A}}]$, s_1 and s_2.
- Outputs
  $(K, \hat{B}, ((X, pk_N), (Y, B, pk_N)), ((B), (X)))$.

If either party outputs $\perp$, it is assumed that both parties abort the protocol and delete all temporary state.

4.3 Comparison

In ntor the initiating party, $\hat{A}$, outputs

$$(K, \hat{B}, (\boldsymbol{v}_0 = (X), \boldsymbol{v}_1 = (Y, B)), (\boldsymbol{u}_0 = (B))).$$

Since the output vector dictates the conditions under which a session is deemed fresh, and freshness is a necessary precondition for security, we can read $\hat{A}$'s output as specifying the scenarios that would definitely compromise an ntor session. Clearly each party must contribute some non-compromised keying material in order for the session to be secure. Consequently we see that the component $\boldsymbol{v}_0$ dictates that the adversary must never partner to the initiator's ephemeral key, and $\boldsymbol{v}_1$ dictates that the adversary must never partner to both B and Y. Likewise, an ntor session cannot possibly be secure if the authenticated party's longterm key was compromised prior to or during the session; and so, $\boldsymbol{u}_0$ requires that the adversary does not partner to B prior to session completion.

In ntrutor the initiating party outputs

$$(K, \hat{B}, ((X, pk_N), (Y, B, pk_N)), ((B), (X))).$$

By a similar reading, we see that the adversary may partner to Y or pk_N at any time, but must not partner to X or B while the session is active. After the session is completed the adversary may partner to any subset (or all) of the DH values provided it does not partner to pk_N, or it may partner to pk_N provided it does not partner to X. Collectively these rules model the claim that ntrutor is secure against the failure of *either* the Diffie-Hellman or the NTRU assumption after session completion, but that it relies on the Diffie-Hellman assumption while the session is active.

It is also worth pointing out that, to achieve better efficiency, we do not rely on one-time signatures to bind s_1 and s_2. See Appendix A for more details.

5 Security

In this section we give an argument for the Definition 4 security of ntrutor in the random oracle model.

Theorem 1. *If there exists an algorithm $\mathcal{A}$ that breaks the security of* ntrutor *when* KDF *is instantiated with a random oracle, then one can construct an algorithm $\mathcal{B}$ that solves the gap Diffie-Hellman problem in $\mathbb{G}$ with non-negligible probability, or breaks the semantic security of* NTRUEncrypt.

Proof. Suppose that Ψ is a fresh ntrutor session owned by party $\hat{P}$ and $\mathsf{Test}(\hat{P}, \Psi)$ does not return $\perp$. The party $\hat{P}$ is necessarily an initiator (by definition of Test), and has output a tuple of the form

$$(K, \hat{B}, ((X, pk_N), (Y, B, pk_N), ((B), (X)))).$$

Since the KDF is modeled as a random oracle, the Test challenge is indistinguishable from a uniform random λ-bit string unless $\mathcal{A}$ has queried the oracle with exactly the same input as $\hat{P}$, specifically[4]:

$$\mathsf{CDH}(X, Y) \mid \mathsf{CDH}(X, B) \mid \mathsf{NTRUDec}(c, sk_N).$$

The algorithm $\mathcal{B}$ is given black-box access to $\mathcal{A}$ and simulates the environment with which $\mathcal{A}$ interacts. $\mathcal{B}$ takes as input a CDH instance (U, V) and an instance of the semantic security game for NTRUEncrypt, specifically a pair of messages m_0, m_1, a public key $\widetilde{pk}$, and a ciphertext $\widetilde{c}$ promised to be an encryption of either m_0 or m_1 under $\widetilde{pk}$.

Let $n = \text{poly}(\lambda)$ be the number of parties $\mathcal{A}$ will initialize in the responder role and let $k_i = \text{poly}(\lambda)$ for $i \in [1, n]$ be the number of sessions in which $\hat{P}_i$ will participate.[5]

The algorithm $\mathcal{B}$ begins by selecting distinct party indices $i, j \in [1, n]$, session indices $\ell \in [1, k_i]$, $m \in [1, k_j]$, and a bit r uniformly at random. We denote by $\hat{P}_1$ and $\hat{P}_2$ the parties indexed by i and j; similarly we let Ψ_1 and Ψ_2 denote the sessions involving $\hat{P}_1$ and $\hat{P}_2$ indexed by ℓ and m respectively.

Having fixed these values $\mathcal{B}$ begins the simulation and handles $\mathcal{A}$'s activation requests in accordance with the ntrutor protocol with the following exceptions[6]:

[4] Here we have rearranged the inputs and omitted public values such as the parties' public keys and the string ntrutor for compactness

[5] We fix these quantities for convenience, $\mathcal{B}$ could search for the correct values with polynomial overhead.

[6] Not included in this list, but still important to note, is that if $r = 1$ then $\mathcal{B}$ does not know $\hat{P}_1$'s long-term secret and is unable to handle any “respond” requests involving $\hat{P}_1$ honestly. However since $\mathcal{B}$ simulates all of the parties it can use the initiator's ephemeral secret to produce s_1 as $X^y|B^x$ and can, otherwise, still follow the protocol in these situations.

1. If $r = 1$, then in response to "init server" request number i, $\mathcal{B}$ registers V as the longterm public key of $\hat{P}_1$.
2. We assume that "start" request number ℓ involving $\hat{P}_1$ is directed at an anonymous party, $\hat{A}_1$ (otherwise $\mathcal{B}$ aborts). In response to this request, $\mathcal{B}$ simulates $\hat{A}_1$ by performing the normal input validation, session creation, and NTRUGen routines, but skips DHGen and inserts U into the outgoing "respond" request in place of an ephemeral DH key.
   If $r = 0$ then $\mathcal{B}$ simulates the response of $\hat{P}_1$ by generating c honestly, selecting K and $auth$ uniformly at random, and inserting V into the outgoing "finish" request instead of the ephemeral DH key.
   If $r = 1$ then $\mathcal{B}$ simulates the response of $\hat{P}_1$ by generating both c and Y honestly, and selecting K and $auth$ uniformly at random.
   Finally $\mathcal{B}$ simulates the response of $\hat{A}_1$ to the "finish" request by outputting
   $$(K, \hat{P}_1, ((U, pk_N), (V, B, pk_N)), ((B), (U)))$$
   in the $r = 0$ case and
   $$(K, \hat{P}_1, ((U, pk_N), (Y, V, pk_N)), ((V), (U)))$$
   in the $r = 1$ case.
3. We assume that "start" request number m involving $\hat{P}_2$ is directed at an anonymous party, $\hat{A}_2$. In response to this request, $\mathcal{B}$ simulates $\hat{A}_2$ by performing the normal input validation, session creation, and DHGen routines, but skips NTRUGen and inserts $\widetilde{pk}$ into the outgoing "respond" request in place of an ephemeral NTRU key.
   $\mathcal{B}$ simulates the response of $\hat{P}_2$ by selecting K and $auth$ uniformly at random, and inserting $\widetilde{c}$ into the "finish" request. The simulated output of $\hat{A}_2$ in response is
   $$(K, \hat{P}_2, ((X, \widetilde{pk}), (Y, B, \widetilde{pk})), ((B), (X))).$$
4. If $\mathcal{B}$ cannot simulate one of $\mathcal{A}$'s activation requests, for instance a Partner query involving U, then $\mathcal{B}$ aborts the simulation.

Suppose that $\mathcal{B}$ has not aborted the simulation and $\mathcal{A}$ queries Test on some session. Since $\mathcal{B}$ chose the sessions to modify uniformly at random, and $\mathcal{A}$ cannot distinguish these sessions from the others, there is a non-negligible probability that $\mathcal{A}$ selects either Ψ_1 or Ψ_2 for its Test query. There are now two cases to consider.

Case 1. $\mathcal{A}$ has queried Test on session Ψ_1. Since $\mathcal{B}$ did not abort, $\mathcal{A}$ has not issued a Partner query for the initiator's ephemeral DH key, and is partner to at most one of the responder's DH keys depending on r. We show that $\mathcal{B}$ can extract a CDH solution from $\mathcal{A}$.

Suppose $r = 0$. The initiator's output is
$$(K, \hat{P}_1, ((U, pk_N), (V, B, pk_N), ((B), (U)))),$$
and since $\mathcal{A}$ has not issued partner requests for U or V it cannot distinguish this output from an honestly generated one. Recall that in the random oracle model $\mathcal{A}$ wins the Test challenge iff it queries
$$\mathsf{CDH}(U, V) \mid \mathsf{CDH}(U, B) \mid \mathsf{NTRUDec}(c, sk_N).$$
$\mathcal{B}$ uses the DDH oracle to recognize this query among all of the those made by $\mathcal{A}$, and in doing so extracts the solution $\mathsf{CDH}(U, V)$ to its input.

Now suppose $r = 1$. The initiator's output is
$$(K, \hat{P}_1, ((U, pk_N), (Y, V, pk_N), ((V), (U)))),$$
and again $\mathcal{A}$ cannot distinguish this output from an honestly generated one. As above, $\mathcal{B}$ is able to extract $\mathsf{CDH}(U, V)$ from $\mathcal{A}$'s random oracle query by checking each query with the DDH oracle.

Case 2. $\mathcal{A}$ has queried Test on session Ψ_2. Since $\mathcal{B}$ did not abort, $\mathcal{A}$ has not issued a Partner query for the initiator's ephemeral NTRUEncrypt key, but may be partner to any or all of the DH values. We show that $\mathcal{B}$ can break the semantic security of NTRUEncrypt.

The initiator's output is
$$(K, \hat{P}_2, ((X, \widetilde{pk}), (Y, B, \widetilde{pk}), ((B), (X)))).$$
Without loss of generality assume $\mathsf{NTRUDec}(\widetilde{c}, \widetilde{sk}) = m_0$. Then $\mathcal{A}$ wins the Test challenge iff it queries
$$\mathsf{CDH}(X, Y) \mid \mathsf{CDH}(Y, B) \mid m_0.$$
Such queries are easily identified and, with all but negligible probability, $\mathcal{A}$ does not make a similar query containing m_1. As such, by examining $\mathcal{A}$'s queries, $\mathcal{B}$ can break the semantic security of NTRUEncrypt.

Recall that we assume $\mathcal{A}$ wins the Test challenge with non-negligible advantage in a non-simulated environment. By the freshness condition it can do so either by partnering to the test session's ephemeral NTRUEncrypt key and at most one of the DH values, or without partnering to the ephemeral NTRUEncrypt key. $\mathcal{A}$ cannot detect when it is in the simulated environment, so its advantage carries over. If it succeeds after partnering to the ephemeral NTRUEncrypt key, then by case 1 above $\mathcal{B}$ solves its GDH instance with non-negligible probability. Otherwise, by case 2, $\mathcal{B}$ breaks the semantic security of NTRUEncrypt. □

5.1 Related security concerns

One-way anonymity The one-way anonymity (as defined in [8]) of our protocol follows immediately from the one-way anonymity of ntor as proven in [8]. Intuitively, the only additional information in an ntrutor transcript is a set of ephemeral NTRUEncrypt public keys, and these are non-identifying. For a full proof, $\mathsf{Partner}(\cdot, pk_N)$ queries must be forbidden in addition to $\mathsf{Partner}(\cdot, X)$ queries.

Forward Secrecy Our protocol clearly meets the criteria for forward secrecy in Definition 5. The responder's certified key, B, is the only long-term value appearing in the protocol, and B is included in a $\boldsymbol{u}$ component of the initiator's output. Furthermore, the adversary who does not partner to X, Y or pk_N may partner to B after session completion without violating freshness.

Quantum resistance Our protocol is not quantum-resistant under Definition 6, as a fully quantum adversary can compute the discrete logarithm of a long-term authentication key and use it to violate the authenticity of new sessions.

Forward quantum-resistance Our protocol is forward quantum-resistant under Definition 7. We model cryptanalytic attacks on DH components as Partner queries. If an adversary uses a quantum computer to solve the relevant CDH instances they are partner to X, Y, and B. This precludes them from partnering to pk_N without violating the freshness of the session. By Theorem 1, since the attacker is not partner to pk_N, violating the security of ntrutor implies breaking the semantic security of NTRUEncrypt. Since this is assumed to be hard even for quantum adversaries, the protocol is forward quantum-resistant.

Multiple Encryptions In [7], Dodis and Katz presented the notion of CCA security of multiple encryptions.

In a multiple encryption setting, one party splits a message into many blocks, and encrypts each block using a (different) ciphersuite. The other party then decrypts those ciphertext and combines the blocks to recover the message. It is observed in [7] that although individual ciphersuites are secure, combining them together may leak information. In such scenarios, depending on the security level, an attacker is given different powers. We provide more details in the Appendix.

Those assumptions are adequate for encryptions since it is quite usual for different encrypted blocks to be transmitted via different links, and assuming that some of the links are compromised is quite natural. While in a key agreement protocol, those settings appear to be unnecessarily strong: the attacker is allowed to query each encryption scheme on its challenge ciphertext, just not at the same time. In other words, the attacker can query PartialReveal on both s_1 and s_2 without compromising the freshness of a session - which violates the 1-AKE freshness of our model.

Nevertheless, we remark that our protocol is weak Multiple CCA secure. Indeed, a proper security model for our protocol with respect to MCCA lies in between weak MCCA and normal MCCA. See the Appendix for more details.

6 Implementation and performance characteristics

	TAP	ntor	ntrutor
client → server bytes	186	84	693
server → client bytes	148	64	673
client computation (stage 1)	280 μs	84 μs	272 μs
server computation	771 μs	263 μs	307 μs
client computation (stage 2)	251 μs	180 μs	223 μs
total computation time	1302 μs	527 μs	802 μs
% client	40.8%	50.1%	61.7%

Table 1. Performance comparison of TAP, ntor, and ntrutor

We have implemented our protocol [18] with curve25519, ntruees439ep1 and sha256 and integrated it into Tor-0.2.5.6-alpha [17]. This parameterization provides an estimated $\lambda = 128$ bit security level against both active classical adversaries and passive quantum adversaries.

Benchmarks comparing our instantiation's performance with that of ntor and that of the legacy Tor handshake (TAP) are presented in Table 1. The data was gathered using Tor's internal benchmarking utility on an Intel Core i7-2640M CPU at 2.80GHz with TurboBoost disabled. RSA and $\mathbb{Z}_p^*$ Diffie-Hellman operations for TAP were provided by OpenSSL 1.0.1i. The elliptic curve DH operations for ntor and ntrutor were performed by the donna_c64 implementation of curve25519 from NaCL-20110221 [2]. The

ntruees439ep1 operations were provided by the NTRU reference code from Security Innovation [24] compiled with SSSE3 support.

7 Conclusion and future work

We have presented a key exchange protocol that satisfies reasonable definitions of security, one-way authenticity, forward secrecy, quantum forward-resistance. We have also demonstrated that the scheme is practical, and compares favorably with protocols that are currently widely deployed.

Our proposal inherits all of the security properties of the original ntor protocol, but also enjoys forward quantum-resistance due to NTRUEncrypt. We leave the development of a protocol satisfying our notion of quantum-resistance (Definition 6), as well as the definition of a model in which disaster-resistance can be considered, to future work. While it would be relatively straightforward to define a quantum-resistant authenticated key exchange protocol using available quantum-safe authentication mechanisms, more research needs to be done into making these mechanisms efficient before a fully quantum-resistant authenticated key exchange protocols are practical.

Finally we note that our scheme does not depend on any feature of NTRUEncrypt other than its semantic security in the presence of a quantum adversary. As such any quantum-safe key encapsulation mechanism could potentially serve as a drop-in replacement for it. While a more modular system may be desirable in other contexts, Tor's principal goal of preserving anonymity is best served by allowing only a single circuit-extension handshake method.

8 Acknowledgements

The authors would like to thank Nick Matthewson, Michele Mosca, and Douglas Stebila for insightful conversations. John Schanck was supported in part by Industry Canada, NSERC, and the Ontario Research Fund (ORF).

References

1. Carlisle M. Adams, Guenther Kramer, Serge Mister, and Robert J. Zuccherato. On the security of key derivation functions. In Kan Zhang and Yuliang Zheng, editors, *ISC*, volume 3225 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 2004.
2. Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. NaCL: Networking and Cryptography Library. http://nacl.cr.yp.to/, 2011.
3. Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem, August 2014.
4. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.
5. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
6. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *USENIX Security Symposium*, pages 303–320. USENIX, 2004.
7. Yevgeniy Dodis and Jonathan Katz. Chosen-ciphertext security of multiple encryption. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 188–209. Springer, 2005.
8. Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Des. Codes Cryptography*, 67(2):245–269, 2013.
9. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.
10. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, volume 7071 in Lecture Notes in Computer Science, pages 19–34. Springer Berlin Heidelberg, January 2011.
11. Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648. Springer, 2010.
12. Kaoru Kurosawa and Yvo Desmedt. A new paradigm of hybrid encryption scheme. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2004.
13. Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.
14. Kristin Lauter and Anton Mityagin. Security analysis of kea authenticated key exchange protocol. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume

3958 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2006.
15. Michele Mosca, Douglas Stebila, and Berkant Ustaoglu. Quantum key distribution in the classical authenticated key exchange framework. In Philippe Gaborit, editor, *Post-Quantum Cryptography*, volume 7932 of *Lecture Notes in Computer Science*, pages 136–154. Springer, 2013.
16. Chris Peikert. Lattice cryptography for the internet. In Michele Mosca editor, *Post-Quantum Cryptography*, volume 8772 of *Lecture Notes in Computer Science*, pages 197–219. Springer, 2014.
17. The Tor Project. Tor's source code, https://gitweb.torproject.org/tor.git, 2014.
18. John Schanck, William Whyte, and Zhenfei Zhang. ntru-tor reference implementation. https://github.com/NTRUOpenSourceProject/ntru-tor, 2014.
19. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS*, pages 124–134. IEEE Computer Society, 1994.
20. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.
21. Victor Shoup. On formal models for secure key exchange, November 1999.
22. Victor Shoup. A proposal for an ISO standard for public key encryption. *IACR Cryptology ePrint Archive*, 2001:112, 2001.
23. Martijn Stam. A Key Encapsulation Mechanism for NTRU. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 410–427. Springer, 2005.
24. William Whyte, Mark Etzel, and Peter Jenney. NTRUOpenSourceProject. https://github.com/NTRUOpenSourceProject/ntru-crypto, 2014.
25. William Whyte and Jeffrey Hoffstein. NTRU. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 858–861. Springer, 2011.

A Multiple Encryption

To suit the definition of multiple encryption, we rewrite our protocol in terms of dual encryption as follows:

- KeyGen: It takes as input a security parameter λ and outputs two key pairs (sk_1, pk_1) and (sk_2, pk_2), where $sk_1 = \{x, y\}$ and $pk_1 = \{g\}$; (sk_2, pk_2) is a NTRUEncrypt key pair.
- Encrypt: It takes as input the public key and a message $M = (m_1, m_2)$, where $m_1 = NULL$ and $m_2 = s_2$ it outputs $C = (c_1, c_2)$ where $c_1 = (g^x, g^y)$ and $c_2 = c$.
- Decrypt: It takes as input the secret key and a ciphertext, it outputs $s_1 = g^{xy}$ and s_2.
- Combine: It takes as input the m_1 and m_2, it outputs a secret seed $s \leftarrow \mathsf{H}(m_1|m_2)$, where H is a cryptographic hash function.

For the multiple encryption, there exists three levels of CCA security: the weak multiple CCA security (wMCCA), the standard multiple CCA security (MCCA) and the strong multiple CCA security (sMCCA). When the attacker is challenged with an ciphertext C, for the first notion, there exists a oracle such that given any $C' \neq C$ it replies with the secret s'; for MCCA, the oracle replies s' as well as m'_1 and m'_2; for the strongest notion, their exist additional oracles such that given c'_i it replies with m'_i. The difference between the last two notions is that in the normal setting, the adversary cannot query individual decryption oracles, i.e., he needs to submit the query in the form of C.

Definition 8 (Multiple CCA Security). *A protocol is weak/standard/strong Multiple CCA (w/-/sMCCA) secure if their is no adversary who can win the following game with a probability more than $\frac{1}{2}+\varepsilon$, where ε is negligible in λ.*

- *For two messages M_0 and M_1, the challenge randomly pick $b \in \{0, 1\}$ and encrypts M_b and obtain a ciphertext C and send M_1, M_2 and C to the adversary;*
- *the adversary has access to the following oracles*
  - *for any input M an encryption oracle $\mathcal{O}_E$ generates corresponding C;*
  - *for any input m_i an encryption oracle $\mathcal{O}_{e_i}$ generates corresponding c_i;*
  - *(wMCCA) for any input $C' \neq C$, a decryption oracle $\mathcal{O}_H$ returns corresponding hashed value s';*
  - *(MCCA) the above oracles, plus for any input $C' \neq C$, a decryption oracle $\mathcal{O}_D$ returns corresponding M';*
  - *(sMCCA) the above oracles, plus, for any input c_i a decryption oracle $\mathcal{O}_{d_i}$ returns m_i;*
- *the attacker outputs b.*

The adversary wins the game if he guess b correctly.

Our protocol satisfies the requirement of wMCCA, since our construction follows the Dodis-Katz framework for wMCCA secure multiple encryption: s_1 and s_2 are transmitted via two different ciphersuits, and a cryptographic hash function is applied to combine them together.

Finally, we argue that a proper model for the dual key exchange lies in between wMCCA and MCCA: the attacker is allowed to compromise most of the ciphersuits, but there exist at least one ciphersuit that remains secure; in the MCCA setting, this means for n different encryption schemes, the attacker is given decryption oracles $\mathcal{O}_{e_i}$ for as many as $n-1$ schemes.