Post-quantum key exchange for the TLS protocol from the ring learning with errors problem

Joppe W. Bos[1], Craig Costello[2], Michael Naehrig[2], and Douglas Stebila[3,*]

[1] NXP Semiconductors, Leuven, Belgium

[2] Microsoft Research, Redmond, Washington, USA

[3] Queensland University of Technology, Brisbane, Australia

joppe.bos@nxp.com, craigco@microsoft.com, mnaehrig@microsoft.com, stebila@qut.edu.au

August 5, 2014

Abstract

Lattice-based cryptographic primitives are believed to offer resilience against attacks by quantum computers. We demonstrate the practicality of post-quantum key exchange by constructing ciphersuites for the Transport Layer Security (TLS) protocol that provide key exchange based on the *ring learning with errors (R-LWE) problem*; we accompany these ciphersuites with a rigorous proof of security. Our approach ties lattice-based key exchange together with traditional authentication using RSA or elliptic curve digital signatures: the post-quantum key exchange provides forward secrecy against future quantum attackers, while authentication can be provided using RSA keys that are issued by today's commercial certificate authorities, smoothing the path to adoption.

Our cryptographically secure implementation, aimed at the 128-bit security level, reveals that the performance price when switching from non-quantum-safe key exchange is not too high. With our R-LWE ciphersuites integrated into the OpenSSL library and using the Apache web server on a 2-core desktop computer, we could serve 506 `RLWE-ECDSA-AES128-GCM-SHA256` HTTPS connections per second for a 10 KiB payload. Compared to elliptic curve Diffie–Hellman, this means an 8 KiB increased handshake size and a reduction in throughput of only 21%. This demonstrates that post-quantum key-exchange can already be considered practical.

*D.S. supported by Australian Research Council (ARC) Discovery Project DP130104304.

Contents

1 Introduction

While lattice-based primitives [Reg05, Reg06] have been used to achieve exciting new cryptographic functionalities like fully homomorphic encryption [Gen09] and multilinear maps [GGH13], there has also been a great deal of work on instantiating traditional cryptographic functionalities using lattices. One of the catalysts for this direction of research is that, unlike other number-theoretic primitives such as RSA [RSA78] and elliptic curve cryptography (ECC) [Mil85, Kob87], lattices are currently believed to offer resilience against attacks using quantum computers. Motivated by such *post-quantum* security, in this work we replace the traditional number-theoretic key exchange in the widely deployed Transport Layer Security (TLS) protocol [DR08] with one based on the *ring learning with errors (R-LWE) problem* [LPR10], which is related to hard lattice problems.

Our basic key exchange protocol is simple—similar to the unauthenticated Diffie–Hellman protocol [DH76]—and comes with a rigorous proof of security based on the R-LWE problem. We put it in the context of TLS by (i) constructing a TLS ciphersuite that uses R-LWE key exchange rather than elliptic curve Diffie–Hellman (ECDH), (ii) providing a proof in a suitable security model [JKSS12] that this new TLS ciphersuite is a secure channel, and (iii) integrating our software implementation into the OpenSSL library to benchmark its performance against elliptic curve Diffie–Hellman key exchange. This analysis gives practitioners an idea of the price one would pay for using R-LWE to cure post-quantum paranoia today. We focus our work on the key exchange component, not authentication: we assume that a quantum computer does not currently exist so that the standard RSA-based authentication in TLS is secure for now. However, using R-LWE as a current key exchange mechanism would assure us that if a quantum computer is realized at some stage in the future, the ephemeral session keys we establish today will remain secure so long as the corresponding R-LWE problem does.

R-LWE at the 128-bit security level. The implementation we describe in this work is intended to serve as a drop-in replacement for traditional forward-secret key exchange mechanisms targeting the 128-bit security level, e.g. in place of the standardized elliptic curve `nistp256`, which is the most widely used elliptic curve in TLS [BHH+13] and provides much faster key exchange than finite-field Diffie–Hellman. While the complexities of the best known attacks against these traditional primitives are widely agreed upon, the state of affairs for attacks against R-LWE is altogether different: for one, there are more parameters that affect the security level in the realm of ideal lattices, so many papers differ significantly in their suggested combinations of parameter sizes for particular security levels. In addition, the majority of authors giving concrete parameters in the (R-)LWE setting have done so at the 80-bit security level. Thus, in this work we have always opted for the conservative approach, making security parameters larger or smaller than they might need to be at stages where the actual attack complexity is unclear. The upshot is that our performance timings could be viewed as somewhat of an upper bound for R-LWE key exchange at the 128-bit level.

Implementation and performance. We implemented the R-LWE algorithms in C. As it has become mandatory to guard cryptographic implementations against physical attacks, such as the leakage of secret material over *side channels* [Koc96], we have implemented an important counter-measure against such attacks by ensuring our implementation has *constant run-time*: the execution time of the implementation does not depend on the input. In practice this is usually realized by eliminating all code that contains data-dependent branches, however this often incurs a performance penalty, so we also provide performance of our variable-time implementation in order to highlight the price one pays in practice when guarding against physical attacks in the context of R-LWE. The most expensive R-LWE operation is sampling from the error distribution, which takes slightly over one million cycles, and has to be performed once by the client and twice by the server during key exchange.

We integrated our R-LWE algorithm into the OpenSSL library. Whereas OpenSSL's implementation of ECDH using the `nistp256` curve takes about 0.8 ms total (here we are referring to just the ECDH point operations, no other signing or encryption operations), the R-LWE operations take about 1.4 ms for the client's operations and 2.1 ms for the server's operations (on a standard desktop computer; see Section 6.2): our constant-time R-LWE implementation is a factor of 1.8–2.6 times slower than ECDH. However, our implementation is entirely in C, so further performance improvements could be achieved through assembly-level optimizations or architecture specific optimization like using the vector instruction set, as well as through

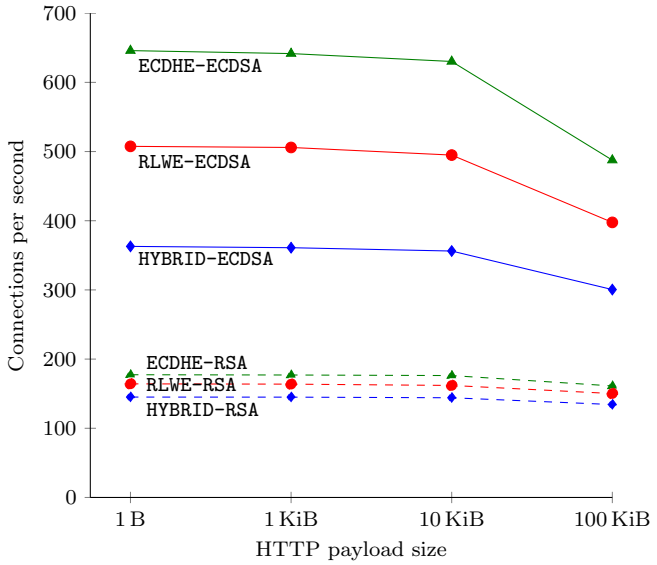


Figure 1: HTTPS connections per second supported by the server at 128-bit security level. All ciphersuites use `AES-128-GCM` and `SHA256`.

a more aggressive parameter selection.

While this is a significant performance difference in terms of raw cryptographic operations, the penalty of using R-LWE becomes less pronounced when used in the context of TLS, as seen in Fig. 1. We did performance testing with our modified OpenSSL in the context of the Apache web server. Using a 3072-bit RSA certificate for authentication (we chose RSA for authentication because the vast majority of commercial certificate authorities only support RSA, and we chose 3072-bit RSA keys to match the desired 128-bit security level [NIS12]), our 2-core web server (serving 10 KiB web pages) could handle 177 `ECDHE-RSA-AES128-GCM-SHA256` connections per second, compared to 164 `RLWE-RSA` connections per second, a factor of about 1.08 difference. Switching to ECDSA-based authentication, we can serve 642 `ECDHE-ECDSA` versus 506 `RLWE-ECDSA` connections per second, which is a larger gap, but which shows that `RLWE-ECDSA` is still highly competitive. Even for *hybrid ciphersuites*, which use both ECDH and R-LWE key exchange (for users who worry about the potential of quantum computers but still need to use ECDH for reasons such as FIPS compliance[1]), performance is reasonable. It should be noted that using R-LWE instead of ECDH increases the handshake by about 8 KiB.

Our performance data demonstrates that R-LWE is a plausible candidate for providing post-quantum key exchange security in TLS. There is a performance penalty for web servers compared to elliptic curve cryptography—a factor of between 1.08–1.27 in our portable C implementation—but this is not too bad. Moreover, performance improvements in the context of R-LWE can be expected as future research is done on parameter choices and scheme designs, new optimizations are developed, and CPU speeds increase.

Related work. A simple unauthenticated key exchange protocol based on the learning with errors (LWE) problem seems to have been folklore for some time. Ding et al. [DL12, §3] present a DH-like protocol based on LWE and give a security proof. Blazy et al. [BCDP13, Fig. 1, 2] describe a similar DH-like protocol based on LWE but without a detailed analysis. Katz and Vaikuntanathan [KV09] build a password-authenticated key exchange protocol from the LWE problem. Whereas the hardness of LWE is related to the shortest vector problem (SVP) on lattices, the R-LWE problem is related to the SVP on ideal lattices, allowing for shorter parameter sizes. Three existing works present key exchange protocols based on R-LWE: Ding et al. [DL12, §4], Fujioka et al. [FSXY13, §5.2], and Peikert [Pei14, §4.1]. All three bear similarities, although differ somewhat in the error correction of the shared secret. Fujioka et al. and Peikert phrase their protocols as *key encapsulation mechanisms (KEMs)* which have passive (IND-CPA) security. To achieve a fully post-quantum authenticated key exchange protocol, Fujioka et al. use standard techniques [FO99] to compile a passively secure KEM

[1] From the FIPS perspective, non-FIPS keying material when XORed or combined in a PRF (like we are doing) is treated as a "constant" that does not negatively affect security or compliance.

into an actively secure KEM which is used to provide authentication in a KEM-KEM approach [FSXY12], whereas Peikert uses a SIGMA-like design [Kra03]. Our unauthenticated DH-like protocol is similar as well, in fact based directly on Peikert's passively secure KEM, but phrased in a DH-like fashion. One of our main differences is that we integrate the R-LWE key exchange directly into TLS, and provide an implementation with performance measurements.

2 Background on ring learning with errors

This section introduces notation and presents the basic background for cryptographic schemes based on the ring learning with errors (R-LWE) problem, which was introduced in [LPR10] (see also [LPR13, Pei14]). Terminology is mostly as in [Pei14].

2.1 Notation

Let $\mathbb{Z}$ be the ring of rational integers, and let $R = \mathbb{Z}[X]/(\Phi_m(X))$ be the ring of integers of the m-th cyclotomic number field, i.e. $\Phi_m \in \mathbb{Z}[X]$ is the m-th cyclotomic polynomial. In this paper, we restrict to the case of m being a power of 2. This means that $\Phi_m(X) = X^n + 1$ for $n = 2^l$, $l > 0$ and $m = 2n$. Let q be an integer modulus and define $R_q = R/qR \cong \mathbb{Z}_q[X]/(X^n + 1)$ with $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. If χ is a probability distribution over R, then $x \xleftarrow{\$} \chi$ denotes sampling $x \in R$ according to χ. If S is a set, then $\mathcal{U}(S)$ denotes the uniform distribution on S, and we denote sampling x uniformly at random from S either with $x \xleftarrow{\$} \mathcal{U}(S)$ or sometimes $x \xleftarrow{\$} S$. If $\mathcal{A}$ is a probabilistic algorithm, $y \xleftarrow{\$} \mathcal{A}(x)$ denotes running $\mathcal{A}$ on input x with randomly chosen coins and assigning the output to y. Different typefaces and cases are used to represent different types of objects: Algorithms (also $\mathcal{A}, \mathcal{B}, \dots$); Queries; `Protocols`, `Schemes`, and `ProtocolMessages`; *variables*; security-notions; and `constants`. $\mathrm{Adv}^{\mathsf{xxx}}_{\mathtt{Y}}(\mathcal{A})$ denotes the advantage of algorithm $\mathcal{A}$ in breaking security notion xxx of scheme or protocol `Y`.

2.2 The decision R-LWE problem

Using the above notation, we define the decision version of the R-LWE problem as follows.

Definition 1 (Decision R-LWE problem)**.** *Let n, R, q and R_q be as above. Let χ be a distribution over R, and let $s \xleftarrow{\$} \chi$. Define $O_{\chi,s}$ as the oracle which does the following:*

1. *Sample $a \xleftarrow{\$} \mathcal{U}(R_q)$, $e \xleftarrow{\$} \chi$,*
2. *Return $(a, as + e) \in R_q \times R_q$.*

The decision R-LWE problem *for n, q, χ is to distinguish $O_{\chi,s}$ from an oracle that returns uniform random samples from $R_q \times R_q$. In particular, if $\mathcal{A}$ is an algorithm, define the advantage*

$$\mathrm{Adv}^{\mathsf{drlwe}}_{n,q,\chi}(\mathcal{A}) = \left| \Pr\left(s \xleftarrow{\$} \chi; \mathcal{A}^{O_{\chi,s}}(\cdot) = 1\right) - \Pr\left(\mathcal{A}^{\mathcal{U}(R_q \times R_q)}(\cdot) = 1\right) \right| \ .$$

Note that the R-LWE problem presented here is stated in its so-called *normal form*, which means that the secret s is chosen from the error distribution instead of the uniform distribution over R_q as originally defined in [LPR10]. See [LPR13, Lemma 2.24] for a proof of the fact that this problem is as hard as the one in which s is chosen uniformly at random.

2.3 Rounding and reconciliation functions

The remainder of this section introduces notation and concepts needed for the key exchange protocols below, mainly following [Pei14]. Let $\lfloor \cdot \rceil : \mathbf{R} \to \mathbb{Z}$ be the usual rounding function, i.e. $\lfloor x \rceil = z$ for $z \in \mathbb{Z}$ and $x \in [z - 1/2, z + 1/2)$.

Definition 2. *Let q be a positive integer. Define the* modular rounding *function*

$$\lfloor \cdot \rceil_{q,2} : \mathbb{Z}_q \to \mathbb{Z}_2, \ x \mapsto \lfloor x \rceil_{q,2} = \left\lfloor \tfrac{2}{q} x \right\rceil \bmod 2.$$

and the cross-rounding *function*

$$\langle\cdot\rangle_{q,2} : \mathbb{Z}_q \to \mathbb{Z}_2,\ x \mapsto \langle x\rangle_{q,2} = \left\lfloor \tfrac{4}{q}x \right\rceil \bmod 2.$$

Both functions are extended to elements of R_q coefficient-wise: for $f = f_{n-1}X^{n-1} + \cdots + f_1X + f_0 \in R_q$, define

$$\begin{aligned} \lfloor f \rceil_{q,2} &= \left(\lfloor f_{n-1}\rceil_{q,2}, \lfloor f_{n-2}\rceil_{q,2}, \ldots, \lfloor f_0\rceil_{q,2}\right), \\ \langle f\rangle_{q,2} &= \left(\langle f_{n-1}\rangle_{q,2}, \langle f_{n-2}\rangle_{q,2}, \ldots, \langle f_0\rangle_{q,2}\right). \end{aligned}$$

In [Pei14], Peikert defines a reconciliation mechanism using the above functions. If the modulus q is odd, it requires to work in $\mathbb{Z}_{2q}$ instead of $\mathbb{Z}_q$ to avoid bias in the derived bits. Since we use odd q in this paper, we need to introduce the randomized doubling function from [Pei14]: let $\mathrm{dbl} : \mathbb{Z}_q \to \mathbb{Z}_{2q},\ x \mapsto \mathrm{dbl}(x) = 2x - e$, where e is sampled from $\{-1, 0, 1\}$ with probabilities $p_{-1} = p_1 = \frac{1}{4}$ and $p_0 = \frac{1}{2}$. The following lemma shows that the rounding of $\mathrm{dbl}(v) \in \mathbb{Z}_{2q}$ for a uniform random element $v \in \mathbb{Z}_q$ is uniform random in $\mathbb{Z}_{2q}$ given its cross-rounding, i.e. $\langle\mathrm{dbl}(v)\rangle_{2q,2}$ hides $\lfloor\mathrm{dbl}(v)\rceil_{2q,2}$.

Lemma 1 ([Pei14, Claim 3.3])**.** *For odd q, if $v \in \mathbb{Z}_q$ is uniformly random and $\overline{v} \xleftarrow{\$} \mathrm{dbl}(v) \in \mathbb{Z}_{2q}$, then $\lfloor\overline{v}\rceil_{2q,2}$ is uniformly random given $\langle\overline{v}\rangle_{2q,2}$.*

The randomized doubling function dbl is extended to elements $f \in R_q$ by applying it to each of its coefficients, resulting in a polynomial in R_{2q}, which in turn can be taken as an input to the rounding functions $\lfloor\cdot\rceil_{2q,2}$ and $\langle\cdot\rangle_{2q,2}$.

In [Pei14], a reconciliation function is defined to recover $\lfloor v\rceil_{q,2}$ from an element $w \in \mathbb{Z}_q$ close to an element $v \in \mathbb{Z}_q$, given only w and the cross-rounding $\langle v\rangle_{q,2}$. We recall the definition from [Pei14] working via $\mathbb{Z}_{2q}$ since the modulus q is odd in this paper. Define the sets $I_0 = \{0, 1, \ldots, \lfloor\frac{q}{2}\rceil - 1\}$ and $I_1 = \{-\lfloor\frac{q}{2}\rceil, \ldots, 1\}$. Let $E = [-\frac{q}{4}, \frac{q}{4})$, then the reconciliation function $\mathrm{rec} : \mathbb{Z}_{2q} \times \mathbb{Z}_2 \to \mathbb{Z}_2$ is defined by

$$\mathrm{rec}(w, b) = \begin{cases} 0, & \text{if } w \in I_b + E \mod 2q, \\ 1, & \text{otherwise.} \end{cases}$$

It is shown in the next lemma that one can recover the rounding $\lfloor\mathrm{dbl}(v)\rceil_{2q,2}$ of a random element $v \in \mathbb{Z}_q$ from an element $w \in \mathbb{Z}_q$ close to v and the cross-rounding $\langle\mathrm{dbl}(v)\rangle_{2q,2}$.

Lemma 2 ([Pei14, Section 3.2])**.** *For odd q, let $v = w + e \in \mathbb{Z}_q$ for $w, e \in \mathbb{Z}_q$ such that $2e \pm 1 \in E \pmod q$. Let $\overline{v} = \mathrm{dbl}(v)$. Then $\mathrm{rec}(2w, \langle\overline{v}\rangle_{2q,2}) = \lfloor\overline{v}\rceil_{2q,2}$.*

Again, reconciliation of a polynomial in R_q is done coefficient-wise using the reconciliation function on $\mathbb{Z}_{2q} \times \mathbb{Z}_2$. Note that Lemma 2 ensures that for two polynomials $v, w \in \mathbf{R}_q$ which are close to each other, i.e. $v = w + e$ for a polynomial e, the polynomial w can be exactly reconciled to $\lfloor\overline{v}\rceil_{2q,2}$ given $\langle\overline{v}\rangle_{2q,2}$ whenever every coefficient $e_i \in \mathbb{Z}_q$ of the difference $e \in \mathbf{R}_q$ satisfies $2e_i \pm 1 \in E \pmod q$. The rounding functions in Definition 2 are trivial to implement. They involve a simple precomputed partitioning of $\mathbb{Z}_q$ into two (not necessarily connected) subdomains D and $\mathbb{Z}_q \setminus D$, and on input of $x \in \mathbb{Z}_q$, these functions return a bit depending on whether $x \in D$ or not. The time taken by the rounding functions is negligible compared to the other cryptographic operations while the time of the doubling function is dominated by sampling of the necessary random bits (see Secton 6.1 for more performance details).

2.4 Discrete Gaussians

The distribution χ referred to in the above definition of the R-LWE problem is usually a discrete Gaussian distribution on R. Since this paper restricts to the case of $m = 2^l$ being a power of 2, sampling from a discrete Gaussian can be done by sampling each coefficient from a 1-dimensional discrete Gaussian $D_{\mathbb{Z},\sigma}$ with parameter σ. The discrete Gaussian (see [DG14]) assigns to each $x \in \mathbb{Z}$ a probability proportional to $e^{-x^2/(2\sigma^2)}$, normalized by the factor $S = 1 + 2\sum_{k=1}^{\infty} e^{-k^2/(2\sigma^2)}$, given by $D_{\mathbb{Z},\sigma}(x) = \frac{1}{S}e^{-x^2/(2\sigma^2)}$. The discrete Gaussian distribution on R is the discrete Gaussian $D_{\mathbb{Z}^n,\sigma}$ obtained by sampling each coefficient from $D_{\mathbb{Z},\sigma}$.

Public parameters		
Decision R-LWE parameters q, n, χ $a \xleftarrow{\$} \mathcal{U}(R_q)$		
Alice		**Bob**
$s, e \xleftarrow{\$} \chi$		$s', e' \xleftarrow{\$} \chi$
$b \leftarrow as + e \in R_q$	$\xrightarrow{b}$	$b' \leftarrow as' + e' \in R_q$
		$e'' \xleftarrow{\$} \chi$
		$v \leftarrow bs' + e'' \in R_q$
		$\overline{v} \xleftarrow{\$} \mathrm{dbl}(v) \in R_{2q}$
	$\xleftarrow{b', c}$	$c \leftarrow \langle \overline{v} \rangle_{2q,2} \in \{0,1\}^n$
$k_A \leftarrow \mathrm{rec}(2b's, c) \in \{0,1\}^n$		$k_B \leftarrow \lfloor \overline{v} \rceil_{2q,2} \in \{0,1\}^n$

Figure 2: Unauthenticated Diffie–Hellman-like key exchange from R-LWE.

3 Unauthenticated Diffie–Hellman-like key exchange protocol

In this section we describe an unauthenticated Diffie–Hellman-like key exchange protocol based on the R-LWE problem. In order to have an exact key exchange protocol, we need to apply error correction to Alice's computation of the shared secret. We employ Peikert's error correction mechanisms described in §2.3, resulting in a key exchange protocol that is effectively a reformulation of Peikert's KEM [Pei14, §4]. This protocol, shown in Fig. 2, is a rephrasing of the following computational problem:

Definition 3 (Exact DDH-like problem)**.** *Let q, n, χ be R-LWE parameters. The* exact decision Diffie–Hellman-like (e-ddh) *problem for q, n, χ is to distinguish DH-like tuples with a real shared secret from those with a random value, given reconciliation information. If $\mathcal{A}$ is an algorithm, define*

$$\mathrm{Adv}^{\text{e-ddh}}_{q,n,\chi}(\mathcal{A}) = \left|\Pr\left(\mathcal{A}(a, b, b', c, k) = 1\right) - \Pr\left(\mathcal{A}(a, b, b', c, k') = 1\right)\right| \ ,$$

where $a \xleftarrow{\$} \mathcal{U}(R_q)$, $s, s', e, e', e'' \xleftarrow{\$} \chi$, $b \leftarrow as + e$, $b' \leftarrow as' + e'$, $v \leftarrow bs' + e''$, $\overline{v} \xleftarrow{\$} \mathrm{dbl}(v)$, $c \leftarrow \langle \overline{v} \rangle_{2q,2}$, $k \leftarrow \lfloor \overline{v} \rceil_{2q,2}$, and $k' \xleftarrow{\$} \mathcal{U}(\{0,1\}^n)$.

Theorem 1 (Hardness of exact DDH-like problem)**.** *Let q be an odd integer, let n be a parameter, and χ be a distribution on R_q. If the decision R-LWE problem for q, n, χ is hard, then the exact DDH-like problem for q, n, χ is also hard. More precisely,*

$$\mathrm{Adv}^{\text{e-ddh}}_{n,q,\chi}(\mathcal{A}) \leq \mathrm{Adv}^{\text{drlwe}}_{n,q,\chi}(\mathcal{A} \circ \mathcal{B}_1) + \mathrm{Adv}^{\text{drlwe}}_{n,q,\chi}(\mathcal{A} \circ \mathcal{B}_2)$$

where $\mathcal{B}_1$ and $\mathcal{B}_2$ are the reduction algorithms given in Fig. 3.

Proof. The proof closely follows Peikert's proof of IND-CPA security of the related KEM [Pei14, Lemma 4.1]. It proceeds by a sequence of games which are shown in Fig. 3. Let S_i be the event that the adversary guesses the bit b^* in Game i.

Game 0. This is the original game, where the messages are generated honestly as in Fig. 2. We want to bound $\Pr(S_0)$. Note that in Game 0, the R-LWE pairs are: (a, b) (with secret s); and (a, b') and (b, v) (both with secret s'). Hence,

$$\mathrm{Adv}^{\text{e-ddh}}_{n,q,\chi}(\mathcal{A}) = \left|\Pr(S_0) - 1/2\right| \ . \tag{1}$$

Game 1. In this game, Alice's ephemeral public key is generated uniformly at random, rather than being generated as a R-LWE sample from distribution χ and public parameter a. Note that in Game 1, the R-LWE pairs are: (a, b') and (b, v) (both with secret s').

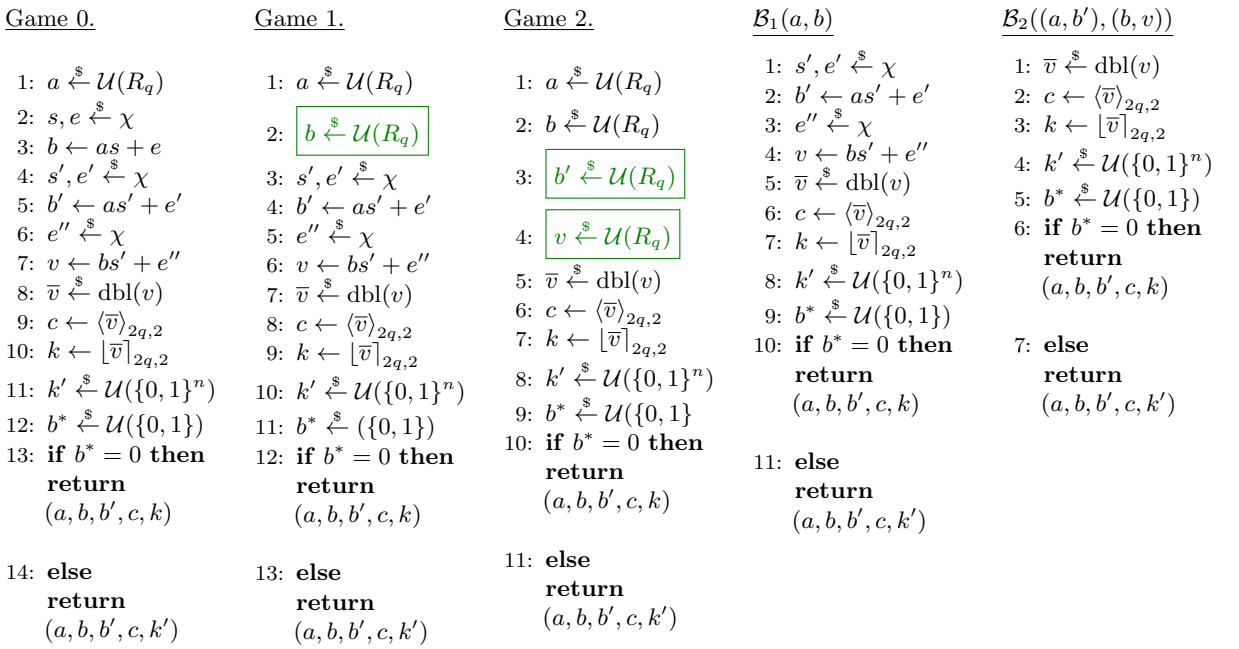


Figure 3: Sequence of games and reductions $\mathcal{B}_1$ and $\mathcal{B}_2$ for proof of Theorem 1.

Difference between Game 0 and Game 1. In Game 0, (a, b) is a sample from $O_{\chi,s}$. In Game 1, (a, b) is a sample from $\mathcal{U}(R_q^2)$. Under the decision ring learning with errors assumption (Definition 1), these two distributions are indistinguishable.

More explicitly, let $\mathcal{B}_1$ be the algorithm shown in Fig. 3 that takes as input a pair (a, b). When (a, b) is a sample from $O_{\chi,s}$ where $s \xleftarrow{\$} \chi$, then the output of $\mathcal{B}_1$ is distributed exactly as in Game 0. When (a, b) is a sample from $\mathcal{U}(R_q^2)$, then the output of $\mathcal{B}_1$ is distributed exactly as in Game 1. Thus, if $\mathcal{A}$ can distinguish Game 0 from Game 1, then $\mathcal{A} \circ \mathcal{B}_1$ can distinguish samples from $O_{\chi,s}$ from samples from $\mathcal{U}(R_q^2)$. Thus,

$$|\Pr(S_0) - \Pr(S_1)| \leq \mathrm{Adv}^{\mathsf{drlwe}}_{n,q,\chi}(\mathcal{A} \circ \mathcal{B}_1) \ . \tag{2}$$

Game 2. In this game, the shared secret key k is generated uniformly at random, rather than being generated via a combination of Alice and Bob's ephemeral keys. Note that in Game 2, there are no R-LWE pairs.

Difference between Game 1 and Game 2. In Game 1, (a, b') and (b', v) are two samples from $O_{\chi,s'}$. In Game 2, (a, b') and (b', v) are two samples from $\mathcal{U}(R_q^2)$. Under the decision ring learning with errors assumption (Definition 1), these two distributions are indistinguishable.

More explicitly, let $\mathcal{B}_2$ be the algorithm shown in Fig. 3 that takes as input two pairs $((a, b'), (b, v))$. When (a, b') and (b, v) are samples from $O_{\chi,s'}$ where $s' \xleftarrow{\$} \chi$, then the output of $\mathcal{B}_2$ is distributed exactly as in Game 1. When (a, b') and (b, v) are samples from $\mathcal{U}(R_q^2)$, then the output of $\mathcal{B}_2$ is distributed exactly as in Game 2. Thus, if $\mathcal{A}$ can distinguish Game 1 from Game 2, then $\mathcal{A} \circ \mathcal{B}_2$ can distinguish samples from $O_{\chi,s}$ from samples from $\mathcal{U}(R_q^2)$. Thus,

$$|\Pr(S_1) - \Pr(S_2)| \leq \mathrm{Adv}^{\mathsf{drlwe}}_{n,q,\chi}(\mathcal{A} \circ \mathcal{B}_2) \ . \tag{3}$$

Analysis of Game 2. In Game 2, the adversary is asked to guess b^* and thereby distinguish between k and k'. Since k is computed as $k \leftarrow \lfloor \overline{v} \rceil_{2q,2}$ where v is chosen uniformly at random from R_q and $\overline{v} \xleftarrow{\$} \mathrm{dbl}(v)$, we have from Lemma 1 that k is distributed uniformly on $\{0,1\}^n$, even given $c = \langle \overline{v} \rangle_{2q,2}$. As well, k' is chosen uniformly at random from $\{0,1\}^n$. Note that k and k' are independent of the values a, b, b', c provided to the

adversary. Thus, the adversary has no information about b^*, and hence

$$\Pr(S_2) = 1/2 \ . \tag{4}$$

Conclusion. Combining equations (1)–(4), we obtain the result in the theorem. □

4 Implementing R-LWE

In this section we describe two implementations of the R-LWE key exchange protocol. The difference between the two implementations arises from the fact that one takes additional measures to ensure that the routine runs in *constant-time*, meaning that there is no data flow from secret material to branch conditions.

4.1 Parameter selection

For our implementation, we chose the following parameters: $n = 1024$, $q = 2^{32} - 1$, $\sigma = 8/\sqrt{2\pi} \approx 3.192$. These parameters provide a security of at least 128 bits against the distinguishing attack described in [MR09, LP11, vdPS13, LN14] with distinguishing advantage less than 2^{-128}. To achieve a certain advantage requires the adversary to find a short vector of a certain length in a corresponding lattice. We evaluated the parameters with the analysis from [LN14], which uses the BKZ-2.0 simulation algorithm according to [CN11]. Based on the simulation results, the parameters guarantee that an adversary running BKZ 2.0 cannot obtain a vector of the required size in 2^{128} steps, keeping the distinguishing advantage below 2^{-128}.

Previous works on implementing lattice-based cryptographic primitives typically use smaller dimension (usually $n = 512$, provided the schemes are not used for homomorphic encryption for which dimensions are much larger). By increasing the dimension to 1024 we are particularly conservative against progress in lattice-basis reduction algorithms. The size of the modulus q provides a large margin for correctness and could possibly be reduced. Note that according to [BLP+13], the form of the modulus does not have an influence on the security of the LWE problem. Assuming that this also holds for R-LWE, we allow the modulus to be composite.

4.2 Sampling from the Gaussian

In this subsection we describe how to sample *small* elements in the ring R_q; this corresponds to the operations denoted as $\xleftarrow{\$} \chi$ in Fig. 2. We use a simple adaptation of the *inversion method*, which independently samples each of the $n = 1024$ coefficients of an R_q-element from a one-dimensional discrete Gaussian. For more details on inversion sampling, and on (R-)LWE-style sampling in general, see [DG14].

For a one-dimensional, discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$ centred at $\mu = 0$ with standard deviation σ, recall from §2.4 the probability of a random variable taking the value $x \in \mathbb{Z}$ is

$$D_{\mathbb{Z},\sigma}(x) = \frac{1}{S}\, e^{-x^2/(2\sigma^2)},$$

where $S = \sum_{k=-\infty}^{\infty} e^{-k^2/(2\sigma^2)}$; in our case, when $\sigma = 8/\sqrt{2\pi}$, we have $S = 8$.

Our adaptation of inversion sampling uses a precomputed lookup table $T = [T[0], \ldots T[51]]$ of size 52, where $T[0] = \lfloor 2^{192} \cdot \frac{1}{S} \rfloor$, where

$$T[i] = \left\lfloor 2^{192} \cdot \left(\frac{1}{S} + 2 \sum_{x=1}^{i} D_{\mathbb{Z},\sigma}(x) \right) \right\rfloor$$

for $i = 1, \ldots, 50$, and where $T[51] = 2^{192}$. Since $S = 8$, note that all table elements are integers in $[2^{189}, 2^{192}]$, and that $T[i+1] > T[i]$ for $i = 0, \ldots, 50$. The sampling of an element $s \in R_q$, denoted $s \xleftarrow{\$} \chi$, is performed as follows. Write $s = \sum_{j=0}^{1023} s_j X^j$. For each $j = 0, \ldots, 1023$, we independently generate a 192-bit integer v_j uniformly at random, and compute the unique integer index $\mathtt{ind}_j \in [0, 50]$ such that $T[\mathtt{ind}_j] \leq v_j < T[\mathtt{ind}_j + 1]$. We then generate one additional random bit to decide the sign $\mathtt{sign}_j \in \{-1, 1\}$, and return the j-th coefficient as $s_j \leftarrow \mathtt{sign}_j \cdot \mathtt{ind}_j$. Note that since every operation of the form $s \xleftarrow{\$} \chi$ requires 1024 random strings of length 192, in total we need 196,608 bits of randomness for each execution

of the operation $\xleftarrow{\$} \chi$. Since the 1024 coefficients of s are sampled independently, the sampling routine is *embarassingly parallelizable*.

As outlined above, our implementation needs a large amount of random data (e.g. we need 24 KiB of random data each time we sample a small ring element). Provided there is enough entropy, this data could be obtained from a true random number generator, such as OpenSSL's `RAND_bytes` function. Obtaining these truly random bytes is slow and unnecessary (from a security perspective). Instead, one can use a pseudo random number generator (PRNG) which is seeded with truly random data whose size is determined by the security parameter. We use this approach conservatively: for each small ring element we need to sample, we re-seed the PRNG by sampling sufficient truly random data (16 bytes for the 128-bit security level) and obtain the subsequent 24 KiB of random data with the help of the PRNG. In our implementation we use AES in counter mode as the PRNG function.

The security (proof) of our protocol requires that the *statistical difference* of our sampling algorithm and the theoretical distribution is less than 2^{-128}. The proposition below shows that this is indeed the case; the accompanying proof uses two lemmas (and the associated notation) from [DG14].

Proposition 1. *Let D'' be the distribution corresponding to the sampling routine described above and let $D_{\mathbb{Z}^n,\sigma}$ be the true discrete Gaussian distribution on $\mathbb{Z}^n$. The statistical difference $\Delta(D'', D_{\mathbb{Z}^n,\sigma})$ of the two distributions is bounded by*

$$\Delta(D'', D_{\mathbb{Z}^n,\sigma}) < 2^{-128}.$$

Proof. In [DG14, Lemma 1], we use $k = 129$, $m = 1024$, $\sigma = 8$ and $c = 1.30872$ to get $\Pr(||\mathbf{v}|| > 42\sigma) < 2^{-129}$ for $\mathbf{v} \leftarrow D_{\mathbb{Z}^n,\sigma}$. Subsequently, we take $t = 42$ and $\epsilon = 2^{-192}$ in [DG14, Lemma 2] to get $\Delta(D'', D_{\mathbb{Z}^n,\sigma}) < 2^{-k} + 2mt\sigma\epsilon = 2^{-129} + 2 \cdot 2^{10} \cdot 42 \cdot (8/\sqrt{2\pi}) \cdot 2^{-192} < 2^{-128}$. Finally, for all $x \in \mathbb{Z}$ with $|x| > 51$, the true probabilities $D_{\mathbb{Z},\sigma}(x)$ in [DG14, Lemma 2] (where they are denoted ρ_x) are such that $D_{\mathbb{Z},\sigma}(x) < 2^{-192}$, meaning that we can zero the approximate probabilities p_x and maintain $|p_x - D_{\mathbb{Z},\sigma}(x)| = |D_{\mathbb{Z},\sigma}(x)| < \epsilon$; this allows the individual samples to be instead taken from $[-51, 51]$, provided we set $T[51]$ is set as 2^{192} so that $\sum_{x=-51}^{51} p_x = 1$. □

We implemented the above sampling routine in two different ways, based on the way that each index $\mathtt{ind}_j$ is retrieved from T (on input of the random 192-bit integer v_j). The first uses a plain binary search which (with overwhelmingly high probability) returns the correct value $\mathtt{ind}$ using 6 (192-bit) integer comparisons for each j. While this approach uses the same number of identical steps each time it is called, it is not *constant-time*. Accessing elements from different parts of the table might require a variable amount of time depending on whether or not this data was already loaded into the cache. Attacks which use this type of information are known as cache-attacks [OST06]. Thus, we also implemented a truly constant-time sampling routine that loads every table element and creates a mask based on whether each accessed element is smaller than the input or not. This requires exactly 51 (constant-time) integer comparisons, which explains the performance difference between these routines (see Section 6).

4.3 Correctness of the scheme

In this subsection we give a brief, heuristic argument as to why the key agreement scheme in Fig. 2 is indeed *exact*. We start by discussing the coefficient sizes of a product of two small error polynomials. By the cyclic nature of reduction modulo $X^n + 1$, it is straightforward to see that if $x = \sum_{i=0}^{n-1} x_i X^i$ and $y = \sum_{i=0}^{n-1} y_i X^i$ are both sampled from χ, then each coefficient z_i of the product $x \cdot y = \sum_{i=0}^{n-1} z_i X^i$ is of the form $z_i = \sum_{j=0}^{n-1} \mathtt{sign}_j \cdot x_j y_j$, for some reordering of the y_j and where $\mathtt{sign}_j \in \{-1, 1\}$. Given that each of the x_j and y_j are sampled independently from the one-dimensional discrete Gaussian $D_{\mathbb{Z},\sigma}$, the variance of each product is $\mathtt{var}(x_j y_j) = \mathtt{var}(x_j)\mathtt{var}(y_j) = \sigma^4$, so the variance of the i-th coefficient z_i is $\mathtt{var}(z_i) = \sum_{j=0}^{n-1} \mathtt{var}(x_j y_j) = n\sigma^4$. By the *central limit theorem*, the individual coefficients of the product xy become (approximately) normally distributed around zero and with standard deviation $\sigma^2\sqrt{n}$.

Refering back to the key exchange algorithm in Fig. 2, we note that the difference between the two keys (prior to any doubling, rounding, cross-rounding or reconciliation) is $v - b's = s'e + e'' - se'$, where e, e', e'', s, s' are all sampled from χ. Using the argument from above, the variance of the i-th coefficient of this difference, $(v - b's)_i$, can be written in terms of the variances of the i-th coefficients in the summand as

$\mathtt{var}((v-b's)_i) = \mathtt{var}((s'e)_i) + \mathtt{var}((e'')_i) + \mathtt{var}((se')_i) = 2\sigma^4 n + \sigma^2$. Thus, prior to the application of Peikert's reconciliation functions, the average differences between each coefficient of the keys is normally distributed around zero, with standard deviation $\sqrt{2\sigma^4 n + \sigma^2} \approx 460 < 2^9$. Lemma 2 tells us that the reconciliation function will only fail if the absolute value of one or more of these coefficients exceeds $q/8 > 2^{28}$. So $q/8$ is over 2^{19} standard deviations away from the average differences between the coefficients of the unreconciled keys, from which the probability of the reconciled keys disagreeing anywhere is clearly negligible.

4.4 Polynomial arithmetic

The arithmetic used in the key agreement scheme is polynomial arithmetic in the cyclotomic ring $R_q = \mathbb{Z}_q[X]/(\Phi_{2^{l+1}}(X))$ where $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ and $\Phi_{2^{l+1}}(X) = X^{2^l} + 1$ is the 2^{l+1}-th cyclotomic polynomial. We assume that 2 is invertible in the ring $\mathbb{Z}_q$ (i.e. q is odd). Multiplying two elements in R_q can be achieved by computing the discrete Fourier transform via fast Fourier transform (FFT) [CT65] algorithms. More specifically, we use the approach from Nussbaumer [Nus80] based on recursive negacyclic convolutions (see [Knu97, Exercise 4.6.4.59] for more details) since this naturally applies to cyclotomic rings for which the degree is a power of 2.

Nussbaumer observed that for any ring $\mathbf{R}$ in which 2 is invertible and whenever $2^l = k \cdot r$ with $k \mid r$ then

$$\mathbf{R}[X]/(X^{2^l} + 1) \cong (\mathbf{R}[Z]/(Z^r + 1))[X]/(Z - X^k),$$

where $Z^{r/k}$ is a $2k$-th root of unity in $\mathbf{R}[Z]/(Z^r + 1)$. Hence, multiplying by powers of the root of unity is computationally cheap since it boils down to shuffling around the polynomial coefficients. This polynomial multiplication method requires $\mathcal{O}(l \log l)$ multiplications in $\mathbf{R}$. Investigation of other asymptotically efficient polynomial multiplication algorithms, such as Schönhage-Strassen multiplication [SS71], is left as future research.

We implemented the version of Nussbaumer's method as outlined by Knuth [Knu97, Exercise 4.6.4.59]. We use $q = 2^{32} - 1$ to define $\mathbb{Z}_q$, note that q is not prime ($q = (2^1 + 1)(2^2 + 1)(2^4 + 1)(2^8 + 1)(2^{16} + 1)$). This choice of the modulus q allows us to compute the modular reduction efficiently. We follow the implementation strategy for the modular arithmetic in [BCHL13]. If a prime modulus q is required one could use the eighth Mersenne prime: $2^{31} - 1$. This also allows efficient modular reduction but we found that an exponent which is a multiple of eight outweighs other positive performance aspects.

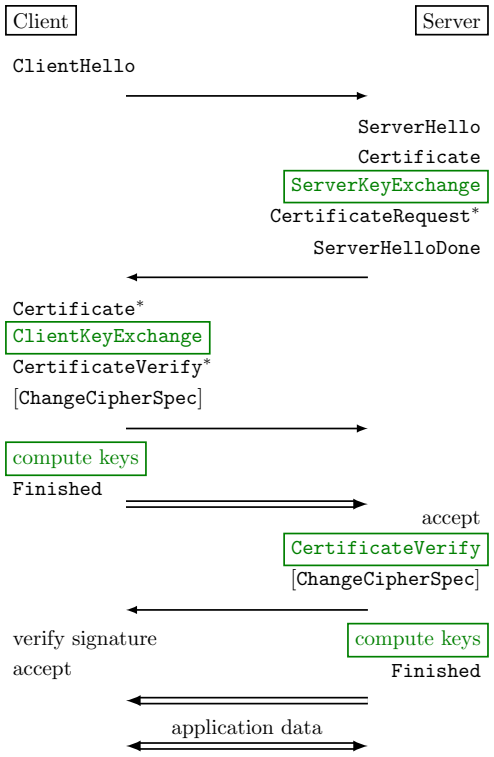
5 Integration into TLS

In this section we discuss the integration of R-LWE into the Transport Layer Security (TLS) protocol. We describe how to integrate the R-LWE-based key exchange protocol into the message flow of TLS, discuss our implementation in the OpenSSL software,[2] and demonstrate that the new ciphersuites satisfies the standard security model for TLS.

5.1 Message flow and operations

The message flow and cryptographic computations for our TLS ciphersuite with signatures for entity authentication and R-LWE for key exchange are given in Fig. 4. The R-LWE key exchange values from the basic protocol (Fig. 2) are inserted in the `ServerKeyExchange` and `ClientKeyExchange` messages. Notice that, compared with signed-Diffie–Hellman ciphersuites, the server's signature is separated out from the `ServerKeyExchange` message into a separate `CertificateVerify` message near the end of the handshake; for the rationale of this design choice, see Remark 4. We fixed a public parameter a to be used by all parties. We generated a once at random; for standardization purposes, a single a value should be generated in a verifiably random, "nothing up my sleeve" manner. The computation of each of the ciphersuite-specific (highlighted) messages, as well as the premaster key derivation, is given in Fig. 4. Notice that, since the server sends the first key exchange message, the server plays the role of Alice from the basic unauthenticated exact Diffie–Hellman-like protocol (Fig. 2) and the client plays the role of Bob.

[2] `http://www.openssl.org/`



`ServerKeyExchange`:

1: $s, e \xleftarrow{\$} \chi$
2: $b \leftarrow as + e$
3: **return** b

`ClientKeyExchange`:

1: $s', e' \xleftarrow{\$} \chi$
2: $b' \leftarrow as' + e'$
3: $e'' \xleftarrow{\$} \chi$
4: $v \leftarrow bs' + e''$
5: $\overline{v} \xleftarrow{\$} \mathrm{dbl}(v)$
6: $c \leftarrow \langle \overline{v} \rangle_{2q,2}$
7: **return** b', c

Client compute keys:

1: $pms \leftarrow \lfloor \overline{v} \rceil_{2q,2}$
2: Compute master secret ms and encryption keys as per TLS specification [DR08, §8.1, §6.3].

Server `CertificateVerify`:

1: Sign handshake messages to this point as in client's `CertificateVerify` algorithm [DR08, §7.4.8].

Server compute keys:

1: $pms \leftarrow \mathrm{rec}(2b's, c)$
2: Compute master secret ms and encryption keys as per TLS specification [DR08, §8.1, §6.3].

* denotes optional messages for client authentication and [`ChangeCipherSpec`] denotes the `ChangeCipherSpec` alert protocol message, after which all data sent by that party is encrypted and authenticated on the record layer. Single lines (⟶) denote unprotected communication, double lines (⟹) denote encrypted/authenticated (record layer) communication, rectangles highlight messages or steps which have changed for this ciphersuite.

Figure 4: The TLS protocol with a signed R-LWE key exchange ciphersuite. Left: TLS message flow. Right: R-LWE-specific computations.

5.2 Implementation

We implemented the R-LWE-based ciphersuite described in the previous subsection into OpenSSL. Specifically, we created four new ciphersuites, all designed to achieve a 128-bit security level. The first two, `RLWE-ECDSA-AES128-GCM-SHA256` and `RLWE-RSA-AES128-GCM-SHA256`, consist of:

- key exchange based on R-LWE key exchange, as described in Section 5.1 and Fig. 4, with parameters as described in Section 4.1, namely $n = 1024$, $q = 2^{32} - 1$, and $\sigma = 8/\sqrt{2\pi}$;
- authentication based on ECDSA or RSA digital signatures;[3]
- authenticated encryption (with associated data) (AEAD) based on AES-128 in GCM (Galois Counter Mode), which provides confidentiality as well as message integrity without the addition of a separate MAC; and
- key derivation and hashing based on SHA-256.

We also created two `HYBRID` ciphersuites that are as above, except the key exchange includes both R-LWE and ECDH key exchange; the pre-master secret is the concatenation of the ECDH shared secret and the R-LWE shared secret.

[3]The ciphersuite does not fix the ECDSA/RSA parameter sizes; to get 128-bit security, we use the `nisp256` curve and 3072-bit RSA signatures.

All these ciphersuite requires TLSv1.2 because of the use of AES-GCM (which we chose since it provably satisfies the stateful length-hiding authenticated encryption notion [PRS11]), but TLSv1.0 ciphersuites using AES in CBC mode are possible.

We integrated our C code implementation of the R-LWE operations described in Section 4 into OpenSSL v1.0.1f. Specifically, we added the unauthenticated exact-DH-like protocol from Section 3 to OpenSSL's `libcrypto` module, then added the TLS ciphersuite to OpenSSL's `libssl` module, and finally extended various OpenSSL command-line programs (such as `openssl speed`, `s_client`, and `s_server`) appropriately. Note that all required randomness is generated using OpenSSL's `RAND_bytes` function. The resulting libraries can then be used with OpenSSL-reliant applications with few or no changes. For example, to use our R-LWE ciphersuite with Apache's `httpd` web server,[4] it suffices to recompile Apache to link against the new OpenSSL library, and set the ciphersuite option in Apache's runtime configuration files. We report on the performance of the new ciphersuite in Section 6.2.

5.3 Security model: authenticated and confidential channel establishment (ACCE)

We now turn to analyzing the security of our new TLS ciphersuite. Jager et al. [JKSS12] introduced the *authenticated and confidential channel establishment (ACCE)* security model to prove the security of TLS. It is based on the Bellare–Rogaway model for authenticated key exchange [BR93], but leaves the key exchange security property implicit, instead having separate properties for entity authentication and channel security, where the latter property is based on the stateful length-hiding authenticated encryption notion introduced by Paterson et al. [PRS11] for the TLS record layer. In this subsection, we present the ACCE security model, and in the next section we prove security of the ciphersuite in that model. Our model differs slightly from the original model in that it explicitly includes forward secrecy. Our presentation is based largely on the text of Dowling et al. [DGK+13].

Parties, long-term keys, and sessions. The **execution environment** consists of n_{P} parties $P_1, \ldots, P_{n_{\mathrm{P}}}$, each of whom is a potential protocol participant. Each party P_i generates a long-term private key / public key pair (sk_i, pk_i). Each party can execute multiple sessions of the protocol, either concurrently or subsequently. We denote the s-th session of a protocol at party P_i by π_i^s, and use n_{S} to denote the maximum number of sessions per party. Each session within the party has read access to the party's long-term key, and read/write access to the per-session variables. We overload notation and use π_i^s to denote the collection of the following per-session variables:

- $\rho \in \{\texttt{init}, \texttt{resp}\}$: The party's role in this session.
- $pid \in \{1, \ldots, n_{\mathrm{P}}, \perp\}$: The identifier of the alleged peer of this session, or $\perp$ for an unauthenticated peer.
- $\alpha \in \{\texttt{inprogress}, \texttt{reject}, \texttt{accept}\}$: The status of the session.
- k: A session key, or $\perp$. Note that k consists of two sub-keys: bi-directional authenticated encryption keys k_e and k_d, which themselves may consist of encryption and possibly MAC sub-keys.
- sid: A session identifier defined by the protocol.
- $st_{\mathrm{E}}, st_{\mathrm{D}}$: State for the stateful authenticated encryption and decryption algorithms (see Definition 9).
- b: A hidden bit (used for a security experiment). On session initialization, $b \xleftarrow{\$} \{0, 1\}$.
- Additional state specific to the security experiment as described in Fig. 5.
- Any additional state specific to the protocol.

Adversary interaction. The adversary controls all communications: it directs parties to initiate sessions, delivers messages to parties, and can reorder, alter, delete, and create messages. The adversary can also compromise certain long-term or per-session secrets. The adversary interacts with the parties using the following queries.

The first query models normal, unencrypted communication of parties during session establishment:

- $\mathsf{Send}(i, s, m) \xrightarrow{\$} m'$: The adversary sends message m to session π_i^s. Party P_i processes message m according to the protocol specification and its per-session state π_i^s, updates its per-session state, and optionally outputs an outgoing message m'. There is a distinguished initialization message which allows the adversary to initiate the session with the role ρ it is meant to play in the session and optionally the

[4] `http://httpd.apache.org/`

$\mathsf{Enc}(i, s, \ell, ad, m_0, m_1)$

1: $\pi_i^s.u \leftarrow \pi_i^s.u + 1$
2: $(C^0, \pi_i^s.st_{\mathrm{E}}^0) \xleftarrow{\$} \mathtt{AENC}.\mathrm{Enc}(\pi_i^s.k, \ell, ad, m_0, \pi_i^s.st_{\mathrm{E}})$
3: $(C^1, \pi_i^s.st_{\mathrm{E}}^1) \xleftarrow{\$} \mathtt{AENC}.\mathrm{Enc}(\pi_i^s.k, \ell, ad, m_1, \pi_i^s.st_{\mathrm{E}})$
4: **if** $C^0 = \perp$ or $C^1 = \perp$ **then return** $\perp$
5: $\pi_i^s.C_{\pi_i^s.u} \leftarrow C^{\pi_i^s.b}$, $\pi_i^s.ad_{\pi_i^s.u} \leftarrow ad$, $\pi_i^s.st_{\mathrm{E}} \leftarrow st_{\mathrm{E}}^{\pi_i^s.b}$
6: **return** $C^{\pi_i^s.b}$

$\mathsf{Dec}(i, s, ad, C)$

1: **if** $\pi_i^s.b = 0$ **then return** $\perp$
2: $j \leftarrow \pi_i^s.pid$, $t \leftarrow$ index of (first) matching session at P_j
3: $\pi_i^s.v \leftarrow \pi_i^s.v + 1$
4: $(m', \pi_i^s.st_{\mathrm{D}}) \leftarrow \mathtt{AENC}.\mathrm{Dec}(\pi_i^s.k, ad, C, \pi_i^s.st_{\mathrm{D}})$
5: **if** $\pi_i^s.v > \pi_j^t.u$ or $c \neq \pi_j^t.C_{\pi_i^s.v}$ or $ad \neq \pi_j^t.ad_{\pi_i^s.v}$ **then**
6: $\quad \pi_i^s.phase \leftarrow 1$
7: **end if**
8: **if** $\pi_i^s.phase = 1$ **then return** m
9: **return** $\perp$

Figure 5: Encrypt and Decrypt oracles for the ACCE security experiment.

identity *pid* of the intended partner of the session. This query may return error symbol $\perp$ if the session has entered state $\alpha = \mathtt{accept}$ and no more protocol messages are transmitted over the unencrypted channel.

The next two queries model adversarial compromise of long-term and per-session secrets:

- $\mathsf{Reveal}(i, s) \rightarrow k$: Returns session key $\pi_i^s.k$.
- $\mathsf{Corrupt}(i) \rightarrow sk$: Return party P_i's long-term secret key sk_i. Note the adversary does not take control of the corrupted party or learn state variables, but can impersonate P_i in later sessions by using sk_i.

The final two queries model communication over the encrypted channel. The adversary can cause plaintexts to be encrypted as outgoing ciphertexts, and can cause ciphertexts to be decrypted and delivered as incoming plaintexts. The queries are used to capture the security notion of stateful length-hiding authenticated encryption as described in Appendix 7.

- $\mathsf{Encrypt}(i, s, m_0, m_1) \xrightarrow{\$} C$: If $\pi_i^s.k = \perp$, the query returns $\perp$. Otherwise, it procceds as in Fig. 5.
- $\mathsf{Decrypt}(i, s, C) \rightarrow m$ or $\perp$: If $\pi_i^s.k = \perp$, the query returns $\perp$. Otherwise, it procceds as in Fig. 5.

The Encrypt/Decrypt oracles, which embody the stateful length-hiding authenticated encryption property, simultaneously capture four security properties:

- *indistinguishability under chosen ciphertext attack*: the adversary cannot distinguish whether m_0 or m_1 is encrypted, even when given access to a decryption oracle;
- *integrity of ciphertexts*: only ciphertexts generated by legitimate parties successfully decrypt;
- *integrity of associated data*; and
- *stateful delivery of ciphertexts*: the adversary cannot deliver ciphertexts out of order.

The hidden bit $\pi_i^s.b$ is leaked to the adversary if any of those conditions are violated.

Security experiment. The ACCE security experiment is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ who implements all parties according to the execution environment above. After the challenger initializes the long-term private key / public key pairs, the adversary receives the public keys and then interacts with the challenger using the queries above. Finally, the adversary outputs a triple (i, s, b') and terminates. The adversary's goal is to either break authentication (by causing an honest session to accept without a matching session) or to successfully guess that the bit b used in the Encrypt/Decrypt oracles of session $SVowner$ is equal to b'. The details of these security goals follow. We begin by defining matching sessions.

Definition 4 (Matching sessions)**.** *We say that session π_j^t* matches *π_i^s if $\pi_i^s.\rho \neq \pi_j^t.\rho$ and $\pi_i^s.sid$ prefix-matches $\pi_j^t.sid$, meaning that (i) if π_i^s sent the last message in $\pi_i^s.sid$, then $\pi_j^t.sid$ is a prefix of $\pi_i^s.sid$, or (ii) if π_j^t sent the last message in $\pi_i^s.sid$, then $\pi_i^s.sid = \pi_j^t.sid$.*

Correctness is defined in the natural way: in the presence of a benign adversary, two communicating oracles will (with overwhelming probability) accept, compute equal session keys, and be able to successfully communicate encrypted application data. For details see [KPW13, full version, Defn. 10].

We can now define server-to-client (a.k.a., server-only) authentication based on the existing of matching sessions.

Definition 5 (Server-to-client authentication)**.** *Let P be a protocol. Let π_i^s be a session. Let $j = \pi_i^s.pid$. We say that π_i^s* accepts maliciously *if*

1. $\pi_i^s.\alpha = \texttt{accept}$*;*
2. $\pi_i^s.\rho = \texttt{init}$*; and*
3. *no* $\mathsf{Corrupt}(j)$ *query was issued before π_i^s accepted,*

but there is no unique session π_j^t which matches π_i^s.

Define $\mathrm{Adv}_P^{\mathsf{acce\text{-}so\text{-}auth}}(\mathcal{A})$ *as the probability that, when $\mathcal{A}$ termiantes in the ACCE experiment for P, there exists a(n initiator) session π_i^s that has accepted maliciously.*

We define channel security based on the adversary's ability to guess the hidden bit b of an uncompromised session, thereby breaking one of the four properties of stateful length-hiding authenticated encryption described above. We focus on channel security in the context of server-only authentication, so the adversary wins if it guesses the hidden bit at any client session, or at any server session in which it was passive.

Definition 6 (Channel security with forward secrecy in server-only authentication mode)**.** *Let P be a protocol. Let π_i^s be a session. Let $j = \pi_i^s.pid$. Suppose $\mathcal{A}$ outputs (i, s, b'). We say that $\mathcal{A}$* answers the encryption challenge correctly *if*

1. $\pi_i^s.\alpha = \texttt{accept}$*;*
2. *no* $\mathsf{Corrupt}(i)$ *query was issued before π_i^s accepted;*
3. *no* $\mathsf{Corrupt}(j)$ *query was issued before any session π_j^t which matches π_i^s accepted;*
4. *no* $\mathsf{Reveal}(i, s)$ *query was ever issued;*
5. *no* $\mathsf{Reveal}(j, t)$ *query was ever issued for any session π_j^t which matches π_i^s;*
6. $\pi_i^s.b = b'$*;*
7. *and either:*
   (a) $\pi_i^s.\rho = \texttt{init}$*; or*
   (b) $\pi_i^s.\rho = \texttt{resp}$ *and there exists a session which matches π_i^s.*

Define $\mathrm{Adv}_P^{\mathsf{acce\text{-}so\text{-}aenc\text{-}fs}}(\mathcal{A})$ *as $|p - 1/2|$, where p is the probability that, in the ACCE experiment for P, $\mathcal{A}$ answers the encryption challenge correctly.*

Remark 1 (Mutual authentication). We focus on the case of server-to-client authentication, as that is the dominant mode in which TLS is used on the Internet. The ACCE framework can deal with the mutual authentication case as well, by removing item 2 from Definition 5 and item 7 from Definition 6.

5.4 Security result

The following informal theorem summarizes our security result for our R-LWE-based TLS ciphersuite. The proof is in the standard model, and does not rely on random oracles.

Theorem 2 (TLS signed R-LWE is a secure ACCE (informal))**.** *Let* `TLS-RLWE-SIG-AENC` *denote the TLS protocol with a R-LWE-based ciphersuite as described in Fig. 4, with* `SIG` *for the signature scheme and* `AENC` *as the stateful length-hiding authenticated encryption for the record layer. Let* `PRF` *denote the pseudorandom function used by TLS in that ciphersuite.*

If the signature scheme `SIG` *is existentially unforgeable under chosen message attack, then* `TLS-RLWE-SIG-AENC` *provides secure server-to-client authentication.*

If additionally the exact-DDH-like problem for the R-LWE parameters is hard, `PRF` *is a secure pseudorandom function, and* `AENC` *is a secure stateful length-hiding authenticated encryption scheme, then* `TLS-RLWE-SIG-AENC` *provides channel security, with forward secrecy, in server-only auth. mode.*

Moreover, the protocol is correct with high probability.

Precise statements for the server-to-client authentication property and the channel security property are given in Lemmas 3 and 4. Precise statements of the correctness property are omitted, as they follow clearly from Section 4.3. Security definitions of standard cryptographic components appear in Appendix 7.

Lemma 3 (Server-to-client authentication)**.** *Let $\mathcal{A}$ denote an adversary against the server-only authentication of* TLS-RLWE-SIG-AENC*. Then, for the reduction algorithm $\mathcal{B}_2$ described in the proof of the lemma,*

$$\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth}}_{\texttt{TLS-RLWE-SIG-AENC}}(\mathcal{A}) \leq \frac{(n_{\mathrm{P}} n_{\mathrm{S}})^2}{2^{\ell_{\mathrm{rand}}}} + n_{\mathrm{P}} \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\texttt{SIG}}(\mathcal{B}_2^{\mathcal{A}}) \ .$$

Proof. The proof proceeds via a sequence of games. Since we have altered the TLS protocol so that the server signs the the whole transcript, our proof is simpler than the signed-DH TLS proof of Jager et al. [JKSS12] or Krawczyk et al. [KPW13]. In particular, our proof follows closely the proof of signed-DH in the Secure Shell (SSH) protocol of Dowling et al. [DGK+13], in which the server signs the whole handshake transcript.

Let break_δ be the event that occurs when a client session accepts maliciously in Game δ in the sense of Definition 5.

Game 0 [original experiment]. This game equals the ACCE security experiment described in Section 5.3. Thus,

$$\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth}}_{\texttt{TLS-RLWE-SIG-AENC}}(\mathcal{A}) = \Pr(\mathsf{break}_0) \ . \tag{5}$$

Game 1 [exclude colliding nonces]. In this game, we add an abort rule for non-unique nonces r_C or r_S. Specifically, the challenger collects a list of all random nonces sampled by the challenger for client or server sessions during the simulation. If one nonce appears on the list twice, the simulator aborts the simulation. This is a transition based on a failure event. There are at most $n_{\mathrm{P}} n_{\mathrm{S}}$ sampled random nonces, each taken uniformly at random from $\{0,1\}^{\ell_{\mathrm{rand}}}$. Thus,

$$\Pr(\mathsf{break}_0) \leq \Pr(\mathsf{break}_1) + \frac{(n_{\mathrm{P}} n_{\mathrm{S}})^2}{2^{\ell_{\mathrm{rand}}}} \ . \tag{6}$$

Game 2 [signature forgery]. In this game, we exclude signature forgeries. Technically, we abort the simulation the first time some session $\pi_{i^*}^{s^*}$ accepts after receiving a signature that was not the output of a session with a matching session identifier and the signing peer's long-term public key was uncorrupted at the time $\pi_{i^*}^{s^*}$ accepted; denote this as event abort_2. We excluded nonce collisions in the previous game, so in this game all values signed by honest parties are different. We show that the abort event is related to a signature forgery.

To demonstrate the signature forgery, we construct an algorithm $\mathcal{B}_2^{\mathcal{A}}$ which simulates the TLS protocol execution as in Game 1. $\mathcal{B}_2$ interacts with $\mathcal{A}$. $\mathcal{B}_2$ receives a public key pk^* from an $\mathsf{euf\text{-}cma}$ signature challenger for $\texttt{SIG}$ and guesses a party j^*. In its simulation of Game 1, $\mathcal{B}_2$ uses pk^* as j^*'s public key and $\mathcal{B}_2$ uses the signing oracle from the $\mathsf{euf\text{-}cma}$ signature challenger for $\texttt{SIG}$ to generate all signatures for j[1]. Under the assumption that the abort event abort_2 occurs, with probability $1/n_{\mathrm{P}}$, session $\pi_{i^*}^{s^*}$ aborted after receiving a forged signature for party j^*. Moreover, the adversary did not issue a $\mathsf{Corrupt}(j^*)$ query. Thus, $\mathcal{B}_2$ can perfectly simulate Game 1: in particular, $\mathcal{B}_2$ can answer any $\mathsf{Corrupt}$ queries for all parties other than j^* without knowing the signing key corresponding to pk^*, and $\mathsf{Corrupt}(j^*)$ is not asked.

Since we have excluded nonce collisions, the signature received by $\pi_{i^*}^{s^*}$ on the handshake transcript of $\pi_{i^*}^{s^*}$ was not output by any instance of j^* that $\mathcal{B}_2$ has simulated. This implies that $\mathcal{B}$ did not query the signing oracle of its $\mathsf{euf\text{-}cma}$ signature challenger for this transcript. This transcript and signature is thus a valid forgery.

Thus, if $\mathcal{B}_2$ guess j^* correctly (which happens with probability $1/n_{\mathrm{P}}$), then $\mathcal{A}$ has helped $\mathcal{B}_2$ find a valid forgery for the $\mathsf{euf\text{-}cma}$ challenger for $\texttt{SIG}$. Thus,

$$\Pr(\mathsf{abort}_2) \leq n_{\mathrm{P}} \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\texttt{SIG}}(\mathcal{B}_2^{\mathcal{A}}) \ . \tag{7}$$

Moreover, games 1 and 2 are indistinguishable as long as the failure event does not occur, so

$$|\Pr(\mathsf{break}_1) - \Pr(\mathsf{break}_2)| \leq \Pr(\mathsf{abort}_2) \ . \tag{8}$$

Game 2 only completes if the abort event abort_2 does not occur. By definition of the abort event, this means that no client session accepts without a matching session whenever the peer's public key was uncompromised. Thus game 2 cannot be won:

$$\Pr(\mathsf{break}_2) = 0 \ . \tag{9}$$

Combining equations (5)–(9) yields the result. □

Lemma 4 (Channel security, server-only auth. mode)**.** *Let $\mathcal{A}$ denote an adversary against the channel security (in server-only authentication mode) of* TLS-RLWE-SIG-AENC *in the sense of Definition 6. Then, for the reduction algorithms $\mathcal{B}_2$ described in the proof of Lemma 3 and $\mathcal{D}_3, \ldots, \mathcal{D}_6$ described in the proof of this lemma,*

$$\begin{aligned}\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}aenc\text{-}fs}}_{\texttt{TLS-RLWE-SIG-AENC}}(\mathcal{A}) \leq \frac{(n_\mathrm{P} n_\mathrm{S})^2}{2^{\ell_{\mathrm{rand}}}} + n_\mathrm{P}\mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\texttt{SIG}}(\mathcal{B}_2^{\mathcal{A}}) \\ + n_\mathrm{P} n_\mathrm{S}\left(\mathrm{Adv}^{\mathsf{e\text{-}ddh}}_{q,n,\chi}(\mathcal{D}_3^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{prf}}_{\texttt{PRF}}(\mathcal{D}_4^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{prf}}_{\texttt{PRF}}(\mathcal{D}_5^{\mathcal{A}}) + \mathrm{Adv}^{\mathsf{slhae}}_{\texttt{AENC}}(\mathcal{D}_6^{\mathcal{A}})\right) \ .\end{aligned}$$

where PRF *is the pseudorandom function used in* TLS-RLWE-SIG-AENC*.*

Proof. The proof again proceeds via a sequence of games, and follows closely the proofs of signed-DH channel security of TLS by Jager et al. [JKSS12] and of SSH by Dowling et al. [DGK+13].

The adversary's goal is to compute that random bit $\pi_{i^*}^{s^*}.b$ of a client session (where the peer's long-term key was uncorrupted at the time the client accepted) or the random bit of a server session (where a matching anonymous client session exists).

Let guess_δ be the event that occurs when $\mathcal{A}$ answers the encryption challenge correctly for session π, namely that $\mathcal{A}$ outputs a tuple (i, s, b') such that $\pi.b = b'$ but all freshness conditions in Definition 6 are satisfied.

Game 0 [original experiment]. This game equals the ACCE security experiment described in Section 5.3. Thus,

$$\mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}aenc\text{-}fs}}_{\texttt{TLS-RLWE-SIG-AENC}}(\mathcal{A}) = |\Pr(\mathsf{guess}_0) - 1/2| \ . \tag{10}$$

Game 1 [exclude non-matching sessions]. In this game, we exclude sessions that have no matching session. Technically, we abort the simulation in either of the following cases:

1. if the adversary's chosen session π is a client session that accepted without a matching session and no $\mathsf{Corrupt}(\pi.pid)$ query occurred before π accepted; or
2. if the adversary's chosen session π is a server session that accepted without a matching session.

In the first case, this directly corresponds to a server impersonation, and thus a break in authentication. The second case is already excluded by Definition 6. Thus,

$$|\Pr(\mathsf{guess}_0) - \Pr(\mathsf{guess}_1)| \leq \mathrm{Adv}^{\mathsf{acce\text{-}so\text{-}auth}}_{\texttt{TLS-RLWE-SIG-AENC}}(\mathcal{A}) \ . \tag{11}$$

Game 2 [guess target session]. In this game, we guess which session will be the adversary's target session. Technically, we pick $(i^*, s^*) \xleftarrow{\$} [n_\mathrm{P}] \times [n_\mathrm{S}]$, then continue as in game 1, and at the end abort if the adversary's chosen session $(i, s) \neq (i^*, s^*)$. Our guess is correct with probability $\frac{1}{n_\mathrm{P} n_\mathrm{S}}$. Thus,

$$\Pr(\mathsf{guess}_1) = n_\mathrm{P} n_\mathrm{S} \Pr(\mathsf{guess}_2) \ . \tag{12}$$

There now exists a unique partner session $\pi_{j^*}^{t^*}$ for the guessed session $\pi_{i^*}^{s^*}$ which can be determined by the simulator by looking for matching client/server nonces.

Game 3 [replace R-LWE premaster secret]. In this game, we replace the premaster secret pms in session $\pi_{i^*}^{s^*}$ and its peer $\pi_{j^*}^{t^*}$ with a value chosen uniformly at random from $\{0,1\}^n$. Any algorithm that can distinguish game 2 from game 3 can be used to construct an algorithm that can distinguish DH-like R-LWE tuples with a real shared secret from those with a random value, in the sense of Definition 3.

More explicitly, let $\mathcal{D}_3^{\mathcal{A}}$ be the following algorithm that receives an exact-DDH-like challenge $(\hat{a}, \hat{b}, \hat{b'}, \hat{c}, \hat{k})$ for R-LWE parameters q, n, χ. $\mathcal{D}_3$ executes just as in game 2 and interacts with $\mathcal{A}$, with the following exceptions.

- The system uses $\hat{a}$ as the global R-LWE parameter a.
- When the target server session (whichever of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is the server) is generating its `ServerKeyExchange` message, the simulator uses the given $\hat{b}$ value, rather than generating b as in Fig. 4.
- When the target client session (whichever of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is the client) is generating its `ClientKeyExchange` message, the simulator uses the given $\hat{b'}$ and $\hat{c}$ values, rather than generating b' and c itself as in Fig. 4.
- When the target client and server sessions are computing keys, the simulator uses $\hat{k}$ as the premaster secret rather than generating pms itself as in Fig. 4.

When $\mathcal{A}$ terminates and outputs (i, s, b'), $\mathcal{D}_3$ outputs b'.

When $\mathcal{D}_3$ receives an exact DDH-like tuple with a real shared secret, then $\mathcal{D}_3$ behaves exactly as in game 2. When $\mathcal{D}_3$ receives an DDH-like tuple with a random shared secret, then $\mathcal{D}_3$ behaves exactly as in game 3. Thus, $\mathcal{D}_3$ behaves differently on real versus random tuples exactly with the same probability that $\mathcal{A}$ behaves differently on game 2 versus game 3:

$$|\Pr(\mathsf{guess}_2) - \Pr(\mathsf{guess}_3)| \leq \mathrm{Adv}_{q,n,\chi}^{\mathsf{e\text{-}ddh}}(\mathcal{D}_3^{\mathcal{A}}) \ . \tag{13}$$

Game 4 [replace master secret]. In this game, we replace the master secret ms in session $\pi_{i^*}^{s^*}$ and its peer $\pi_{j^*}^{t^*}$ with a value chosen uniformly at random from $\{0,1\}^{\ell_{ms}}$, rather than being computed as $ms = \mathtt{PRF}(pms, label_1 \| r_C \| r_S)$, where `PRF` is the pseudorandom function used in TLS, $label_1$ is a fixed string, and r_C and r_S are the client and server random nonces from the `ClientHello` and `ServerHello` messages.

Due to the substitution in the previous game, the premaster secret pms input to `PRF` is chosen uniformly at random from $\{0,1\}^n$. Thus, any algorithm that can distinguish game 3 from game 4 can be used to construct an algorithm that can distinguish the output of `PRF` from random, in the sense of Definition 8.

More explicitly, let $\mathcal{D}_4^{\mathcal{A}}$ be the following algorithm that interacts with a prf challenger for `PRF` as in Definition 8. $\mathcal{D}_4$ executes just as in game 3 and interacts with $\mathcal{A}$, with the following exceptions.

- When the target client session is computing keys, rather than computing ms itself, the simulator outputs the string $label_1 \| r_C \| r_S$ to the prf challenger, which proceeds as in Definition 8, then activates the simulator with a real-or-random output K which the simulator uses as ms.
- When the target server session is computing keys, the simulator uses the same ms as the target client session.

When $\mathcal{A}$ terminates and outputs (i, s, b'), $\mathcal{D}_4$ outputs b'.

When $\mathcal{D}_4$ receives the real PRF result from the prf challenger, $\mathcal{D}_4$ behaves exactly as in game 3. When $\mathcal{D}_4$ receives a random value from the prf challenger, $\mathcal{D}_4$ behaves exactly as in game 4. Thus, $\mathcal{D}_4$ behaves differently on real versus random PRF values exactly with the same probability that $\mathcal{A}$ behaves differently on game 3 versus game 4:

$$|\Pr(\mathsf{guess}_3) - \Pr(\mathsf{guess}_4)| \leq \mathrm{Adv}_{\mathtt{PRF}}^{\mathsf{prf}}(\mathcal{D}_4^{\mathcal{A}}) \ . \tag{14}$$

Game 5 [replace encryption keys]. In this game, we replace the encryption keys k in session $\pi_{i^*}^{s^*}$ and its peer $\pi_{j^*}^{t^*}$ with a value chosen uniformly at random from $\{0,1\}^{\ell_k}$, rather than being computed as $k = \mathtt{PRF}(ms, label_2 \| r_C \| r_S)$, where $label_1$ is a fixed string, and r_C and r_S are the client and server random nonces from the `ClientHello` and `ServerHello` messages.

Due to the substitution in the previous game, the master secret ms input to `PRF` is chosen uniformly at random from $\{0,1\}^{\ell_{ms}}$. Thus, any algorithm that can distinguish game 4 from game 5 can be used to construct an algorithm that can distinguish the output of `PRF` from random, in the sense of Definition 8.

More explicitly, let $\mathcal{D}_5^{\mathcal{A}}$ be the following algorithm that interacts with a prf challenger for `PRF` as in Definition 8. $\mathcal{D}_5$ executes just as in game 4 and interacts with $\mathcal{A}$, with the following exceptions.

- When the target client session is computing keys, rather than computing k itself, the simulator outputs the string $label_2 \| r_C \| r_S$ to the prf challenger, which proceeds as in Definition 8, then activates the simulator with a real-or-random output K which the simulator uses as k.
- When the target server session is computing keys, the simulator uses the same k as in the target client session.

When $\mathcal{A}$ terminates and outputs (i, s, b'), $\mathcal{D}_5$ outputs b'.

When $\mathcal{D}_5$ receives the real PRF result from the prf challenger, $\mathcal{D}_5$ behaves exactly as in game 4. When $\mathcal{D}_5$ receives a random value from the prf challenger, $\mathcal{D}_5$ behaves exactly as in game 5. Thus, $\mathcal{D}_5$ behaves differently on real versus random PRF values exactly with the same probability that $\mathcal{A}$ behaves differently on game 4 versus game 5:

$$|\Pr(\mathsf{guess}_4) - \Pr(\mathsf{guess}_5)| \leq \mathrm{Adv}^{\mathsf{prf}}_{\mathtt{PRF}}(\mathcal{D}_5^{\mathcal{A}}) \ . \tag{15}$$

Analysis of game 5. In game 5, the encryption key k of the target session is information-theoretically independent from the key exchange messages. Thus, any adversary that can break the channel security of the target session can be used to break the underlying stateful length-hiding authenticated encryption scheme $\mathtt{AENC}$.

More explicitly, let $\mathcal{D}_6^{\mathcal{A}}$ be the following algorithm that interacts with a slhae challenger for $\mathtt{AENC}$; recalling Definition 9, this means that the slhae challenger has chosen a secret key, and provides $\mathcal{D}_6^{\mathcal{A}}$ with oracle access to Enc and Dec oracles as in Fig. 6. $\mathcal{D}_6$ executies just as in game 5 and interacts with $\mathcal{A}$, with the following exceptions.

- When $\mathcal{A}$ makes an $\mathsf{Encrypt}(i^*, s^*, \ell, ad, m_0, m_1)$ query to $\mathcal{D}_6$, $\mathcal{D}_6$ makes an $\mathsf{Enc}(\ell, ad, m_0, m_1)$ query to its slhae challenger and returns the result to $\mathcal{A}$.
- When $\mathcal{A}$ makes a $\mathsf{Decrypt}(i^*, s^*, ad, C)$ query to $\mathcal{D}_6$, $\mathcal{D}_6$ makes a $\mathsf{Dec}(ad, C)$ query to its slhae challenger and returns the result to $\mathcal{A}$.

When $\mathcal{A}$ terminates and outputs (i, s, b'), $\mathcal{D}_6$ outputs b'. The challenge bit $\pi_{i^*}^{s^*}.b$ in $\mathcal{D}_6$ corresponds to the challenge bit b in the slhae challenger.

The values generated by $\mathcal{D}_6$ are distributed identically as in game 5. Moreover, $\mathcal{A}$'s guess of b' direcly corresponds to a guess of b' in the slhae experiment. Thus,

$$\Pr(\mathsf{guess}_5) = \mathrm{Adv}^{\mathsf{slhae}}_{\mathtt{AENC}}(\mathcal{D}_6^{\mathcal{A}}) \ . \tag{16}$$

Combining equations (10)–(16) yields the result. □

Remark 2 (Quantum-safe reduction and long-term security). Song [Son14] notes that security proofs of allegedly post-quantum classical schemes typically assume classical adversaries, and it does not immediately follow that the proof "lifts" to provide security against quantum adversaries. Song gives conditions under which a classical proof can be lifted to provide quantum security. Some technical conditions must be met, including that the reduction is a "straight-line" reduction, meaning that the reduction runs the adversary from beginning to end, without rewinding or restarting. Our reductions are straight-line reductions. Thus, it seems that Song's framework should apply: if all of the other primitives in our ciphersuite are quantum-safe with proofs against classical adversaries, then they should also be secure against quantum adversaries.

Even if our non-quantum-safe digital signatures are used in our construction (as we do in our implementation), users still have a long-term security property [MSU13]: a polynomial-time quantum computer built in the future may be able to break authentication of sessions that occur after it exists, but cannot decrypt sessions that were executed before it was active.

Remark 3 (Multi-ciphersuite security). Dowling et al. [DGK+13] extend the ACCE definition to consider the case of multi-ciphersuite security, when long-term public keys are shared across multiple ciphersuites using the same long-term authentication algorithm but different key exchange or encryption algorithms. Just because a ciphersuite is ACCE-secure on its own does not mean that ciphersuite is secure in when its long-term public key is used in other ciphersuites.

Dowling et al. give a framework for proving multi-ciphersuite security. They give a composition theorem that says many mutually compatible ciphersuites are secure with shared long-term keys provided each ciphersuite is ACCE-secure with an auxiliary oracle that provides access to operations based on the long-term

key. One must then prove that the individual ciphersuite remains ACCE-secure even when the auxiliary oracle provides operations based on the long-term key.

The signed finite field and elliptic Diffie–Hellman ciphersuites in TLS do not satisfy this property because the data structure that is signed in these ciphersuites consists just of the random nonces and the ephemeral public key. But Dowling et al. do show that signed-Diffie–Hellman ciphersuites in SSH are multi-ciphersuite secure.

However, in our TLS-R-LWE ciphersuite, the data structure that is signed consists of the entire transcript, which uniquely identifies the ciphersuite. This suffices to be able to prove TLS-R-LWE is ACCE-secure with an auxiliary signing oracle, where the predicate Φ (in Dowling et al.'s framework) is based on the ciphersuite chosen by the server in the `ServerHello` message in the transcript; thus our TLS-R-LWE ciphersuite is multi-ciphersuite secure. It is safe to reuse the same long-term signing with other compatible multi-ciphersuite secure ACCE protocols, including signed-Diffie–Hellman ciphersuites in SSH and a hypothetical future version of signed-DH ciphersuites in TLS that sign the entire transcript.

Remark 4 (Oracle assumptions and moving the server signature). The security proofs of signed-DH ciphersuites in TLS [JKSS12, KPW13] required a new Diffie–Hellman assumption, the PRF-Oracle-Diffie–Hellman assumption. Instead of the normal decision Diffie–Hellman assumption, which assumes that the adversary cannot distinguish real DH tuples (g, g^u, g^v, g^{uv}) from random tuples (g, g^u, g^v, g^w), the PRF-ODF assumption assumes the adversary cannot distinguish tuples of the form $(g, g^u, g^v, \mathrm{F}(g^{uv}, m))$ from $(g, g^u, g^v, z \xleftarrow{\$} \{0,1\}^\ell)$, where m is chosen in advance by the adversary and F is a pseudorandom function (see Appendix 7), even given access to an oracle that outputs $\mathrm{F}(X^v, m')$ for any $X \neq g^u$. While controversial when initially proposed by Jager et al. [JKSS12], Krawczyk et al. [KPW13, full version, Appendix C] later demonstrated that the PRF-ODH assumption was in fact necessary, and that a simple PRF assumption would not suffice.

The reason signed-DH ciphersuites in TLS require the PRF-ODH assumption is that the server's signature comes very early in the protocol (as part of the `ServerKeyExchange`). This signature is only over the client and server nonces and the server's ephemeral public key value; server-to-client authentication of the full handshake transcript is done using a MAC under the session key, which was derived from the DH shared secret. An attacker trying to trick the client into accepting a fake transcript could do so either by forging a signature early in the handshake or by trying to break the session key and MAC calculation later in the handshake. This is why the PRF-oracle-DH assumption is required.

In our R-LWE-based ciphersuite in Fig. 4, we move the server's signature to later in the handshake, so that server-to-client authentication of the full handshake transcript is done using the signature scheme. This allows us to prove server-to-client authentication using just signature security, rather than some oracle-DH-like assumption. Our change however is not just for convenience. As noted by Peikert [Pei14, §5.3], R-LWE assumptions are not hard in an oracle setting: "the reason is related to the search/decision equivalence for (ring)-LWE: the adversary can query the [...] oracle on a specially crafted [values] for which the [...] oracle input is one of only a small number of possibilities (and depends on only a small portion of the secret key), and can thereby learn the entire secret key very easily." Technically, the oracle-like assumption used in TLS would only require security against a single query per secret, whereas the attack discusses the use of multiple queries. It is an interesting open question to determine whether oracle R-LWE assumptions with a single query remain secure.

6 Performance

In this section we outline the performance of the separate components used for the cryptographic implementation, as well as overall performance numbers when integrated within the OpenSSL framework. Timings reported involved two computers. Our "client" computer had an Intel Core i5 (4570R) processor with 4 cores running at 2.7 GHz each. Our "server" computer had an Intel Core 2 Duo (E6550) processor with 2 cores running at 2.33 GHz each. Software was compiled for the `x86_64` architecture with `-O3` optimizations using `llvm` 5.1 (`clang` 503.0.40) on the client computer and `gcc` 4.7.2 on the server computer.

Table 1: Average cycle count of standalone cryptographic operations (on client computer)

Operation	Cycles	
	constant-time	non-constant-time
sample $\xleftarrow{\$} \chi$	1 042 700	668 000
FFT multiplication	342 800	—
FFT addition	1 660	—
dbl$(\cdot)$ and crossrounding $\langle\cdot\rangle_{2q,2}$	23 500	21 300
rounding $\lfloor\cdot\rceil_{2q,2}$	5 500	3,700
reconciliation rec$(\cdot,\cdot)$	14 400	6 800

6.1 Standalone cryptographic operations

The average performance numbers, expressed in cycles on the client computer, for the individual components used in our implementation are summarized in Table 1. We distinguish between operations which have a constant or variable running time. The operation with the highest running time is the sampling. As outlined in Section 4.2, our approach requires a significant amount of random data plus as well as a number of (constant-time) comparison operations to a 52-entry look-up table. Querying this random data consumes most of the time in the sampling function. Accessing all elements in the sampling table in order – as performed for the constant-time approach (see Section 4.2) – can be done relatively efficiently, since the total size is slightly over 1.2 KiB, and this fits in the cache. Although we access $52/6 \approx 8.7$ times more table elements compared to the binary search approach, the overall slow-down is less than a factor two for the sampling functionality. The time for computing the polynomial multiplication using the Nussbaumer FFT algorithm (see Section 4.4) includes the two forward and single inverse FFT transforms. Interestingly, computing the high-degree polynomial multiplications is (much) faster than sampling.

6.2 Within TLS and HTTPS

In this section we draw a comparison between the performance of RSA-signed elliptic curve Diffie–Hellman and RSA-signed R-LWE-based TLS ciphersuites within the context of an HTTPS connection. Our approach for analyzing performance of ECDH versus R-LWE in TLS/HTTPS follows that of Gupta et al. [GSF+04], who analyzed the performance of RSA versus ECDH. Our comparison takes place at the 128-bit security level: for elliptic curve operations we used the `nistp256` curve [Nat99] and for R-LWE operations we used the R-LWE parameters as described in Section 4.1 ($n = 1024$, $q = 2^{32} - 1$, and $\sigma = 8/\sqrt{2\pi}$). For RSA authentication, the server used a 3072-bit RSA self-signed certificate or a `nistp256` ECDSA certificate depending on the ciphersuite; no client authentication was used. As noted in Section 5.2, the implementation is based on OpenSSL v1.0.1f.

OpenSSL cryptographic primitive performance. Table 2 shows the runtime of operations within the context of OpenSSL's crypto library using the `openssl speed` command. For R-LWE, we report the performance of both our constant-time and non-constant-time implementations. OpenSSL's RSA and `nistp256` code is constant-time.

Table 2 shows that, in the context of R-LWE key exchange, a constant-time implementation only slows the client and server down by factors 1.4x and 1.2x (respectively) over a non-constant-time implementation; this presents a strong argument for adopting a sampling routine that is side-channel resistant. When comparing the total runtime for ephemeral key exchange, it is encouraging to see that, on both the client and the server sides, the post-quantum R-LWE key exchange incurs less than a factor 2 performance loss over key exchange using the `nistp256` curve: the client is slower by a factor 1.8x, while the server is only slower by a factor 1.5x.

OpenSSL/Apache TLS performance. Table 3 shows the performance of ECDH, R-LWE, and hybrid ciphersuites within the context of HTTP connections over TLS. For R-LWE, we use the constant-time implementation. The server was running Apache `httpd` 2.4.10 with the prefork module for multi-threading. The client and server computers were connected over a local corporate LAN with less than 1 ms ping time.

Table 2: Average runtime in milliseconds of cryptographic operations using `openssl speed`

Operation	**Client constant-time**	**Server constant-time**	**Client non-constant-time**	**Server non-constant-time**
R-LWE key generation	0.9	1.7	0.6	1.3
R-LWE Bob shared secret	0.5	(1.1)	0.4	(0.9)
R-LWE Alice shared secret	(0.1)	0.4	(0.1)	0.4
Total R-LWE runtime	1.4	2.1	1.0	1.7
EC point multiplication, `nistp256`	0.4	0.7	—	—
Total ECDH runtime	0.8	1.4	—	—
RSA sign, 3072-bit key	(3.7)	8.8	—	—
RSA verify, 3072-bit key	0.1	(0.2)	—	—

Numbers in parentheses are reported for completeness, but do not contribute to the runtime in the client and server's role in the TLS protocol.

Table 3: Performance of HTTPS using Apache with OpenSSL

	ECDHE		RLWE		HYBRID	
	ECDSA	RSA	ECDSA	RSA	ECDSA	RSA
Connections / second						
— 1 B payload	645.9	177.4	507.5	164.2	362.9	145.1
— 1 KiB payload	641.6	177.0	505.9	163.8	361.0	145.0
— 10 KiB payload	630.2	176.2	494.9	161.9	356.2	144.1
— 100 KiB payload	487.6	161.2	397.6	150.2	300.5	134.3
Connection time (ms)	6.0	14.0	45.6	54.0	47.2	54.6
Handshake (bytes)	1 278	2 360	9 469	10 479	9 607	10 690

The first section of Table 3 reports the number of simultaneous connections supported by the server. Multiple client connections were generated using the `http_load` tool (version 09jul2014),[5] which makes many HTTP connections in parallel using OpenSSL for TLS. The client and network configuration was sufficient to ensure that the server's 2 cores had at least 95% utilization during all tests. Session resumption was disabled. To simulate a variety of web page sizes, we ran separate benchmarks where the HTTP payload was 1 byte, 1 KiB = 1024 bytes, 10 KiB, and 100 KiB. Each test was run for 100 seconds; figures reported are the average of 5 runs. The second section of Table 3 reports the time required for the client to establish a connection, measured using Wireshark from when the client opens the TCP connection to the server's IP address to when the client starts to receive the first packet of application data. The final section of the table shows the size of the handshake in each case.

Table 3 shows that, when ECDSA is used as the authentication mechanism, employing R-LWE as the TLS key exchange mechanism achieves between a factor 1.2–1.3x fewer HTTP connections per second than when ECDH key exchange is used. On the other hand, when coupled instead with RSA signatures, the relative difference between the R-LWE and ECDH key exchange components is diluted by the slower authentication, and the number of connections per second is (relatively speaking) much closer. These is a larger ratio when comparing the connection times obtained using ECDH key exchange versus R-LWE key exchange, which may be explained due to the difference in the size of the TLS handshake. In all cases, Table 3 shows that the hybrid version (which combines R-LWE for post-quantum assurance and ECDH for FIPS compliance) naturally performs the slowest. Again however, when coupled with RSA signatures, the number of connections per second is only a factor 1.2x fewer than an ECDH-only connection.

[5] http://www.acme.com/software/http_load/

7 Conclusions

The ring learning with errors (R-LWE) problem is a promising cryptographic primitive that is believed to be resistant to attacks by quantum computers. The decision R-LWE problem naturally leads to a Diffie–Hellman-like unauthenticated key exchange protocol. We have integrated this key exchange mechanism into the Transport Layer Security protocol. The resulting provably secure construction provides post-quantum forward secrecy yet remains practical, both in terms of efficiency and in terms of its integration with the widely-deployed RSA-based public key authentication infrastructure. Our constant-time C implementation in the OpenSSL library shows that web servers using R-LWE key exchange incur a small performance penalty to achieve post-quantum assurance. Even hybrid key exchange—using both R-LWE and elliptic curve Diffie–Hellman for "best of both worlds" security—provides reasonable performance. With post-quantum cryptography still in its early days, future work includes optimization of parameter sizes, implementations, and comparisons between post-quantum primitives.

References

[BCDP13] Olivier Blazy, Céline Chevalier, Léo Ducas, and Jiaxin Pan. Errorless smooth projective hash function based on LWE. Cryptology ePrint Archive, Report 2013/821, 2013. http://eprint.iacr.org/2013/821.

[BCHL13] Joppe W. Bos, Craig Costello, Hüseyin Hisil, and Kristin Lauter. Fast cryptography in genus 2. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 194–210. Springer, May 2013.

[BHH+13] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. Elliptic curve cryptography in practice. Cryptology ePrint Archive, Report 2013/734, 2013. http://eprint.iacr.org/2013/734.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, August 1993.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT*, pages 1–20, 2011.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[DG14] Nagarjun C. Dwarakanath and Steven D. Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Appl. Algebra Eng. Commun. Comput.*, 25(3):159–180, 2014.

[DGK+13] Benjamin Dowling, Florian Giesen, Florian Kohlar, Jörg Schwenk, and Douglas Stebila. Multi-ciphersuite security and the SSH protocol. Cryptology ePrint Archive, Report 2013/813, 2013. http://eprint.iacr.org/2013/813.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DL12] Jintai Ding and Xiaodong Lin. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688, 2012. http://eprint.iacr.org/2012/688.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In Hideki Imai and Yuliang Zheng, editors, *PKC'99*, volume 1560 of *LNCS*, pages 53–68. Springer, March 1999.

[FSXY12] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 467–484. Springer, May 2012.

[FSXY13] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Practical and post-quantum authenticated key exchange from one-way secure key encapsulation mechanism. In Kefei Chen, Qi Xie, Weidong Qiu, Ninghui Li, and Wen-Guey Tzeng, editors, *ASIACCS 13*, pages 83–94. ACM Press, May 2013.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, May 2013.

[GSF+04] Vipul Gupta, Douglas Stebila, Stephen Fung, Sheueling Chang Shantz, Nils Gura, and Hans Eberle. Speeding up secure web transactions using elliptic curve cryptography. In *NDSS 2004*. The Internet Society, February 2004.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293. Springer, August 2012.

[Knu97] Donald E. Knuth. *Seminumerical Algorithms*. The Art of Computer Programming. Addison-Wesley, Reading, Massachusetts, USA, 3rd edition, 1997.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, August 1996.

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 429–448. Springer, August 2013.

[Kra03] Hugo Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer, August 2003.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 636–652. Springer, December 2009.

[LN14] Tancrède Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In David Pointcheval and Damien Vergnaud, editors, *AFRICACRYPT*, volume 8469 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2014.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA*, pages 319–339, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, May 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, May 2013.

[Mil85] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer, August 1985.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer Berlin Heidelberg, 2009.

[MSU13] Michele Mosca, Douglas Stebila, and Berkant Ustaoglu. Quantum key distribution in the classical authenticated key exchange framework. In Philippe Gaborit, editor, *Proc. 5th International Conference on Post-Quantum Cryptography (PQCrypto) 2013*, volume 7932 of *LNCS*, pages 136–154. Springer, 2013. Full version available at http://eprint.iacr.org/2012/361.

[Nat99] National Institute of Standards and Technology. Recommended elliptic curves for Federal government use, July 1999.

[NIS12] NIST. Recommendations for key management – Part 1: General (revision 3), July 2012.

[Nus80] Henri J. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(2):205–215, 1980.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 1–20. Springer, February 2006.

[Pei14] Chris Peikert. Lattice cryptography for the internet. In Michele Mosca, editor, *Proc. 6th International Conference on Post-Quantum Cryptography (PQCrypto) 2014*, LNCS. Springer, 2014. To appear. Full version available at http://eprint.iacr.org/2014/070.

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 372–389. Springer, December 2011.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Reg06] Oded Regev. Lattice-based cryptography (invited talk). In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 131–141. Springer, August 2006.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[Son14] Fang Song. A note on quantum security for post-quantum cryptography. In Michele Mosca, editor, *Proc. 6th International Conference on Post-Quantum Cryptography (PQCrypto) 2014*, LNCS. Springer, 2014. To appear.

[SS71] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.

[vdPS13] Joop van de Pol and Nigel P. Smart. Estimating key sizes for high dimensional lattice-based systems. In *IMA Int. Conf.*, pages 290–303, 2013.

A Additional cryptographic definitions

Definition 7 (Digital signature scheme)**.** *A* digital signature scheme Σ *is a typle of algorithms:*

- $\mathrm{KeyGen}() \xrightarrow{\$} (sk, pk)$*: A probabilistic key generation that generates a secret signing key* sk *and public verification key* pk*.*
- $\mathrm{Sign}(sk, m) \xrightarrow{\$} \sigma$*: A probabilistic signing algorithm that takes as input a signing key* sk *and a message* $m \in \{0,1\}^*$*, and outputs a signature* σ*.*
- $\mathrm{Ver}(pk, m, \sigma) \to \{0,1\}$*: A deterministic verification algorithm that takes as input a verification key* pk*, message* m*, and alleged signature* σ*, and outputs* 0 *or* 1*.*

For an adversary $\mathcal{A}$*, we define its advantage in the existential unforgeability under chosen message attack experiment as*

$$\mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\Sigma}(\mathcal{A}) = \Pr\left(\Sigma.\mathrm{Ver}(pk, m^*, \sigma^*) = 1 : (sk, pk) \xleftarrow{\$} \Sigma.\mathrm{KeyGen}() \ ; \ (m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\Sigma.\mathrm{Sign}(sk,\cdot)}(pk)\right)$$

with the restriction that $\mathcal{A}$ *never queries* $\Sigma.\mathrm{Sign}(sk, \cdot)$ *on input* m^**.*

Our definition of a pseudorandom function and a stateful length-hiding authenticated encryption scheme follows that of [KPW13, full version, p. 43–45].

Definition 8 (Pseudorandom function)**.** *A* pseudorandom function F *with key space* $\{0,1\}^{\lambda_1}$ *and input space* $\{0,1\}^*$ *is a deterministic algorithm. On input a key* $k \in \{0,1\}^{\lambda_1}$ *and an input string* $x \in \{0,1\}^*$*, the algorithm outputs a value* $\mathrm{F}(k, x) \in \{0,1\}^{\lambda_2}$*.*

For a stateful adversary $\mathcal{A}$*, we define the PRF distinguishing advantage for* F *as*

$$\mathrm{Adv}^{\mathsf{prf}}_{\mathrm{F}}(\mathcal{A}) = \Pr\left(b' = b : k \xleftarrow{\$} \{0,1\}^{\lambda_1} \ ; \ x \xleftarrow{\$} \mathcal{A}^{\mathrm{F}(k,\cdot)}() \ ; \ k_0 \leftarrow \mathrm{F}(k, x) \ ; \ k_1 \xleftarrow{\$} \{0,1\}^{\lambda_2} \ ; \ b \xleftarrow{\$} \{0,1\} \ ; \ b' \xleftarrow{\$} \mathcal{A}^{\mathrm{F}(k,\cdot)}(k_b)\right) .$$

with the restriction that $\mathcal{A}$ *never queries* $\mathrm{F}(k, \cdot)$ *on input* x*.*

Definition 9 (Stateful length-hiding authenticated encryption)**.** *A* stateful length-hiding authenticated encryption scheme Π *is a tuple of algorithms:*

- $\mathrm{Gen}() \xrightarrow{\$} K$*: A probabilistic key generation algorithm that chooses a key* K *at random from the keyspace* $\mathcal{K}$ *and outputs it.*

$\underline{\mathrm{Exp}^{\mathsf{slhae}}_{\Pi}(\mathcal{A})}$

1: $K \xleftarrow{\$} \Pi.\mathrm{Gen}()$
2: $(st^0_{\mathrm{E}}, st^0_{\mathrm{D}}) \leftarrow \Pi.\mathrm{Init}()$
3: $i \leftarrow 0$; $j \leftarrow 0$; $phase \leftarrow 0$
4: $b \xleftarrow{\$} \{0,1\}$
5: $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec}}()$
6: **if** $b' = b$ **then return** 1
7: **else return** 0

$\underline{\mathsf{Enc}(\ell, ad, m_0, m_1)}$

1: $i \leftarrow i + 1$
2: $(C^0, st^0_{\mathrm{E}}) \xleftarrow{\$} \Pi.\mathrm{Enc}(K, \ell, ad, m_0, st_{\mathrm{E}})$
3: $(C^1, st^1_{\mathrm{E}}) \xleftarrow{\$} \Pi.\mathrm{Enc}(K, \ell, ad, m_1, st_{\mathrm{E}})$
4: **if** $C^0 = \perp$ or $C^1 = \perp$ **then return** $\perp$
5: $C_i \leftarrow C^b$, $ad_i \leftarrow ad$, $st_{\mathrm{E}} \leftarrow st^b_{\mathrm{E}}$
6: **return** C_i

$\underline{\mathsf{Dec}(ad, c)}$

1: **if** $b = 0$ **then return** $\perp$
2: $j \leftarrow j + 1$
3: $(m', st_{\mathrm{D}}) \leftarrow \Pi.\mathrm{Dec}(K, ad, C, st_{\mathrm{D}})$
4: **if** $j > i$ or $C \neq C_j$ or $ad \neq ad_j$ **then** $phase \leftarrow 1$
5: **if** $phase = 1$ **then return** m
6: **return** $\perp$

Figure 6: Security experiment for stateful length-hiding authenticated encryption of scheme Π.

- $\mathrm{Init}() \to (st_{\mathrm{E}}, st_{\mathrm{D}})$*: A deterministic initialization algorithm that outputs initial encryption and decryption states* st_{E} *and* st_{D}*.*
- $\mathrm{Enc}(K, \ell, ad, m, st_{\mathrm{E}}) \xrightarrow{\$} (c, st'_{\mathrm{E}})$*: A probabilistic encryption algorithm that takes as input a key* K*, length* $\ell \in \mathbb{N}$*, associated data* $ad \in \{0,1\}^*$*, message* $m \in \{0,1\}^*$*, and encryption state* st_{E}*, and outputs a ciphertext* $c \in \{0,1\}^*$ *or error symbol* $\perp$ *and an updated encryption state* st'_{E}*, where* $|c| = \ell$ *if* $c \neq \perp$*.*
- $\mathrm{Dec}(K, ad, C, st_{\mathrm{D}}) \to (m', st'_{\mathrm{D}})$*: A deterministic decryption algorithm that takes as input a key* K*, associated data* ad*, ciphertext* c*, and decryption state* st_{D}*, and outputs a message* $m' \in \{0,1\}^*$ *or error symbol* $\perp$ *and an updated decryption state* st'_{D}*.*

Correctness is defined in the natural way and is omitted; see Jager et al. [JKSS12] or Krawczyk et al. [KPW13]. For a stateful adversary $\mathcal{A}$*, we define the advantage*

$$\mathrm{Adv}^{\mathsf{slhae}}_{\Pi}(\mathcal{A}) = \Pr\left(\mathrm{Exp}^{\mathsf{slhae}}_{\Pi}(\mathcal{A}) = 1\right)$$

where $\mathrm{Exp}^{\mathsf{slhae}}_{\Pi}(\mathcal{A})$ *is the experiment defined in Figure 6.*