ETSI TR 103 570 V1.1.1 (2017-10)

CYBER;
Quantum-Safe Key Exchanges

Reference
DTR/CYBER-QSC-007

Keywords
algorithm, confidentiality, quantum cryptography, security

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Cyber Security (CYBER).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document compares a selection of proposals for quantum-safe key exchanges taken from the academic literature. In particular, it includes key exchanges based on the Learning with Errors (LWE), Ring-LWE and Supersingular Isogeny Diffie-Hellman (SIDH) problems, as well as key exchanges constructed from the Niederreiter and NTRU key transport schemes.

The present document gives an overview of each key exchange, lists proposed parameters and gives software performance estimates on a range of processors. It also discusses various security and implementation considerations such as active attacks and side-channel vulnerabilities.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

user with regard to a particular subject area.

[i.1] ETSI QKD GS 002: "Quantum Key Distribution (QKD); Use cases".

[i.2] ETSI GR QSC 001: "Quantum-Safe Cryptography (QSC); Quantum-safe algorithmic framework".

[i.3] IETF draft-ietf-tls-tls13-19: "The Transport Layer Security (TLS) protocol version 1.3", 10 March 2017.

[i.4] IETF RFC 7296: "Internet Key Exchange protocol version 2 (IKEv2)", October 2014.

[i.5] ETSI GR QSC 003: "Quantum Safe Cryptography; Case Studies and Deployment Scenarios".

[i.6] I. Biehl, B. Meyer and V. Müller: "Differential fault attacks on elliptic curve cryptosystems" in CRYPTO, 2000.

[i.7] E. Fujisaki and T. Okamoto: "Secure integration of asymmetric and symmetric encryption schemes" in CRYPTO, 1999.

[i.8] E. E. Targhi and D. Unruh: "Post-quantum security of the Fujisaki-Okamoto and OAEP transforms" in TCC, 2016.

[i.9] P. Kocher: "Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems" in CRYPTO, 1996.

[i.10] P. Kocher, J. Jaffe and B. Jun: "Differential power analysis" in CRYPTO, 1999.

[i.11] D. Brumley and D. Boneh: "Remote timing attacks are practical" Computer Networks, vol. 48, no. 5, pp. 701-716, 2005.

[i.12] J. Großschädl, E. Oswald, D. Page and M. Tunstall: "Side-channel analysis of cryptographic software via early-terminating multiplications" in ISIC, 2009.

[i.13] S. Mangard, E. Oswald and T. Popp: "Power analysis attacks: Revealing the secrets of smart cards", New York: Springer Science & Business Media, 2008.

[i.14] J. D. Golić and C. Tymen: "Multiplicative masking and power analysis of AES" in CHES, 2002.

[i.15] M. Rivain and E. Prouff: "Provably secure higher-order masking of AES" in CHES, 2010.

[i.16] M. Ajtai: "Generating hard instances of lattice problems" in STOC, 1996.

[i.17] M. Ajtai and C. Dwork: "A public-key cryptosystem with worst-case/average-case equivalence" in STOC, 1997.

[i.18] J. Hoffstein, J. Pipher and J. H. Silverman: "NTRU: A ring-based public key cryptosystem" in ANTS III, 1998.

[i.19] O. Regev: "On lattices, learning with errors, random linear codes, and cryptography" in STOC, 2005.

[i.20] D. Stehlé, R. Steinfeld, K. Tanaka and K. Xagawa: "Efficient public key encryption based on ideal lattices" in ASIACRYPT, 2009.

[i.21] V. Lyubashevsky, C. Peikert and O. Regev: "On ideal lattices and learning with errors over rings", Journal of the ACM (JACM), vol. 60, no. 6, p. 43, 2013.

[i.22] C. Peikert: "Some recent progress in lattice-based cryptography" in TCC, 2009.

[i.23] J. Ding, X. Xie and X. Lin: "A simple provably secure key exchange scheme based on the Learning with Errors problem", IACR ePrint Archive 2012/688, 2012.

[i.24] C. Peikert: "Lattice cryptography for the internet" in PQC, 2014.

[i.25] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan and D. Stebila: "Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE", IACR ePrint Archive 2016/659, 2016.

[i.26] S. Battacharya, O. Garcia-Morchon, R. Rietman and L. Tolhuizen: "spKEX: An optimized lattice-based key exchange", IACR ePrint Archive 2017/709, 2017.

[i.27] E. Alkim, L. Ducas, T. Pöppelmann and P. Schwabe: "Post-quantum key exchange - A new hope" in USENIX Security, 2016.

[i.28] J. Bos, C. Costello, M. Naehrig and D. Stebila: "Post-quantum key exchange for the TLS protocol from the Ring Learning with Errors problem" in Security and Privacy, 2015.

[i.29] E. Alkim, P. Jukabeit and P. Schwabe: "A new hope on ARM Cortex-M", IACR ePrint Archive 2016/758, 2016.

[i.30] S. Fluhrer: "Cryptanalysis of Ring-LWE based key exchange with key share reuse", IACR ePrint Archive 2016/085, 2016.

[i.31] J. Ding, S. Alsayigh, R. V. Saraswathy and S. Fluhrer: "Leakage of signal function with reused keys in RLWE key exchange", IACR ePrint Archive 2016/1176, 2016.

[i.32] P. Hodgers, F. Regazzoni, R. Gilmore, C. Moore and T. Ode: "State-of-the-art in physical side-channel attacks and resistant technologies", SAFECrypto D7.1, 2016.

[i.33] L. Groot Bruinderink, A. Hülsing, T. Lange and Y. Yaro: "Flush, Gauss and reload - A cache attack on the BLISS lattice-based signature scheme" in CHES, 2016. .

[i.34] T. Espitau, P. A. Fouque, B. Gerard and M. Tibouch: "Side-channel attacks on BLISS lattice-based signatures", IACR ePrint Archive 2017/505, 2017.

[i.35] R. Primas, P. P. and S. Mangar: "Single-trace side-channel attacks on masked lattice-based encryption", IACR ePrint Archive 2017/594, 2017.

[i.36] S. Roy, O. Reparaz, F. Vercauteren and I. Verbauwhed: "Compact and side channel secure discrete Gaussian sampling", IACR ePrint Archive 2014/591, 2014.

[i.37] O. Reparaz, S. Roy, F. Vercauteren and I. Verbauwhede: "A masked Ring-LWE implementation" in CHES, 2015.

[i.38] T. Oder, T. Schneider, T. Pöppelmann and T. Güneys: "Practical CCA2-secure and masked Ring-LWE implementation", IACR ePrint Archive 2016/1109, 2016.

[i.39] V. Sing: "A practical key exchange for the internet using lattice cryptography", IACR ePrint Archive 138/2015, 2015.

[i.40] V. Singh and A. Chopr: "Even more practical key exchanges for the internet using lattice cryptography", IACR ePrint Archive 2015/1120, 2015.

[i.41] M. R. Albrecht, R. Player and S. Scot: "On the concrete hardness of learning with errors", Journal of Mathematical Cryptology, vol. 9, no. 3, pp. 169-203, 2015.

[i.42] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler and D. Stehlé: "CRYSTALS - Dilithium: Digital signatures from module lattices", IACR ePrint Archive 2017/634, 2017.

[i.43] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck and D. Stehlé: "CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM", IACR ePrint Archive 2017/634, 2017.

[i.44] A. Langlois and D. Stehlé: "Worst-case to average-case reductions for module lattices", Designs, Codes and Cryptography, vol. 75, no. 3, pp. 565-599, 2015.

[i.45] D. Jao and L. De Feo: "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies" in PQC, 2011.

[i.46] L. De Feo, D. Jao and J. Plût: "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies", Journal of Mathematical Cryptography, vol. 8, no. 3, pp. 209-247, 2014.

[i.47] C. Costello, P. Longa and P. Naehrig: "Efficient algorithms for supersingular isogeny Diffie-Hellman" in CRYPTO, 2016.

[i.48] S. D. Galbraith, C. Petit, B. Shani and Y. B. Ti: "On the security of supersingular isogeny cryptosystems" in ASIACRYPT, 2016.

[i.49] S. D. Galbraith and F. Vercauteren: "Computational problems in supersingular elliptic curve isogenies", IACR ePrint Archive 2017/774, 2017.

[i.50] D. Kirkwood, B. Lackey, J. McVey, M. Motley, J. Solinas and D. Tuller: "Failure is not an option: Standardisation issues for post-quantum key agreement" in NIST Workshop on Cybersecurity in a Post-Quantum World, 2015.

[i.51] Y. B. Ti: "Fault attacks on supersingular isogeny cryptosystems" in PQC, 2017.

[i.52] A. Gélin and B. Wesolowski: "Loop-abort faults on supersingular isogeny cryptosystems" in PQC, 2017.

[i.53] C. Costello, D. Jao, P. Longa, M. Naehrig, J. Renes and D. Urbanik: "Efficient compression of SIDH public keys" in EUROCRYPT, 2017.

[i.54] A. Jalali, R. Azarderakhsh, M. Mozaffari-Kermani and D. Jao: "Supersingular isogeny Diffie-Hellman key exchange on 64-bit ARM", IEEE Transactions on Dependable and Secure Computing, vol. (to appear), 2017.

[i.55] B. Koziel, A. Jalali, R. Azarderakhsh, D. Jao and M. Mozaffari-Kermani: "NEON-SIDH: Efficient implementation of supersingular isogeny Diffie-Hellman key exchange protocol on ARM" in CANS, 2016.

[i.56] H. Niederreiter: "Knapsack-type cryptosystems and algebraic coding theory", Problems of Control and Information Theory, vol. 15, no. 2, pp. 159-166, 1986.

[i.57] R. McEliece: "A public key cryptosystem based on algebraic coding theory", DSN progress report 42.44, 1978.

[i.58] V. Shoup: "Fast construction of irreducible polynomials over finite fields", Journal of Symbolic Computation, vol. 17, no. 5, pp. 371-391, 1994.

[i.59] N. Patterson: "The algebraic decoding of Goppa codes", IEEE Transactions on Information Theory, vol. 21, no. 2, pp. 203-207, 1975.

[i.60] E. R. Berlekamp: "Algebraic coding theory", New York: McGraw-Hill, 1968.

[i.61] K. Kobara and H. Imai: "Semantically secure McEliece public-key cryptosystems - conversions for McEliece PKC" in PKC, 2001.

[i.62] E. Persichetti: "Secure and anonymous hybrid encryption from coding theory" in PQC, 2013.

[i.63] Q. Guo, T. Johansson and P. Stankovski: "A key recovery attack on MDPC with CCA security using decoding errors" in ASIACRYPT, 2016.

[i.64] S. Heyse, A. Moradi and C. Paar: "Practical power analysis attakcs on software implementations of McEliece" in PQC, 2010.

[i.65] R. Avanzi, S. Hoerder, D. Page and M. Tunstall: "Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems", Journal of Cryptographic Engineering, vol. 1, no. 4, pp. 271-281, 2011.

[i.66] D. J. Bernstein, T. Chou and P. Schwabe: "McBits: Fast constant-time code-based cryptography" in CHES, 2013.

[i.67] M. Georgieva and F. de Portzamparc: "Toward secure implementation of McEliece decryption" in COSADE, 2015.

[i.68] F. Strenzke: "Timing attacks against the synrome inversion in code-based cryptosystems" in PQC, 2013.

[i.69] PQCrypto: "Initial recommendations of long-term secure post-quantum systems", 2015.

NOTE: Available at http://pqcrypto.eu.org/.

[i.70] D. J. Bernstein, T. Lange and C. Peters: "Attacking and defending the McEliece cryptosystem" in PQC, 2008.

[i.71] S. H. S. de Vries: "Achieving 128-bit security against quantum attacks in OpenVPN", MSc Thesis, University of Twente, 2016.

[i.72] D. J. Bernstein: "List decoding for binary Goppa codes" in International Conference on Coding and Cryptology, 2011.

[i.73] D. J. Bernstein: "Grover vs McEliece" in PQC, 2010.

[i.74] D. J. Bernstein and T. Lange: "eBACS: ECRYPT benchmarking of cryptographic systems".

NOTE: Available at https://bench.cr.yp.to.

[i.75] B. Biswas and N. Sendrier: "McEliece cryptosystem implementation: Theory and practice" in PQC, 2008.

[i.76] T. Chou: "McBits revisited" IACR ePrint Archive 2017/793, 2017.

[i.77] S. Heyse: "Low-reiter: Niederreiter encryption scheme for embedded microcontrollers" in PQC, 2010.

[i.78] W. Whyte: "EEES#1: Implementation aspects of NTRUEncrypt, Version 3.1", 2015.

NOTE: Available at https://github.com/NTRUOpenSourceProject/ntru-crypto/blob/master/doc/EESS1-v3.1.pdf.

[i.79] J. Hoffstein; J. Pipher, J. M. Schanck, J. H. Silverman, W. Whyte and Z. Zhang: "Choosing parameters for NTRUEncrypt" in CT-RSA, 2017.

[i.80] É. Jaulmes and A. Joux: "A chosen-ciphertext attack against NTRU" in CRYPTO, 2000.

[i.81] N. Howgrave-Graham, P. Nguyen, D. Pointcheval, J. Proos, J. Silverman and A. Singer: "The impact of decryption failures on the security of NTRU encryption" in CRYPTO, 2003.

[i.82] N. Gama and P. Q. Nguyen: "New chosen-ciphertext attacks on NTRU" in PKC, 2007.

[i.83] A. Hülsing, J. Rijnveld, J. Schanck and P. Schwabe: "High-speed key encapsulation from NTRU", IACR ePrint Archive 2017/667, 2017.

[i.84] N. Howgrave-Graham, J. Silverman, A. Singer and W. Whyte: "NAEP: Provable security in the presence of decryption failures", IACR ePrint Archive 2003/172, 2003.

[i.85] M. Stam: "A key encapsulation mechanism for NTRU" in Cryptography and Coding, 2005.

[i.86] J. H. Silverman and W. Whyte: "Timing attacks on NTRUEncrypt via variation in the number of hash calls" in CT-RSA, 2007.

[i.87] A. Atici, L. Batina, B. Grierlichs and I. Verbauwhede: "Power analysis on NTRU implementations for RFIDs: First results" in RFIDSec, 2008.

[i.88] A. Wang, X. Zheng and Z. Wang: "Power analysis attacks and countermeasures on NTRU-based wireless body area networks", KSII Transactions on Internet and Information Systems, vol. 7, no. 5, pp. 1094-1107, 2013.

[i.89] X. Zheng, A. Wang and W. Wei: "First-order collision attack on protected NTRU cryptosystem", Microprocessors and Microsystems, vol. 37, pp. 601-609, 2013.

[i.90] N. Howgrave-Graham: "A hybrid lattice-reduction and meet-in-the-middle attack against NTRU" in CRYPTO, 2007.

[i.91] O. M. Guillen, T. Pöppelmann, J. M. B. Mera, E. F. Bongenaar, G. Sigl and J. Sepulveda: "Towards post-quantum security for IoT endpoints with NTRU" in DATE, 2017.

[i.92] J. H. Cheon, J. Jeong and C. Lee: "An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low-level encoding of zero" in ANTS-XII, 2016.

[i.93] M. Albrecht, S. Bai and L. Ducas: "A subfield lattice attack on overstretched NTRU assumptions" in CRYPTO, 2016.

[i.94] D. J. Bernstein, C. Chuengsatiansup, T. Lange and C. van Vredendaal: "NTRU Prime", IACR ePrint Archive 2016/461, 2016.

[i.95] P. Kirchner and P.-A. Fouque: "Comparison between subfield and straightforward attacks on NTRU", IACR ePrint Archive 2016/717, 2016.

[i.96] B. Applebaum, D. Cash, C. Peikert and A. Sahai: "Fast cryptographic primitives and circular-secure encryption based on hard learning problems" in CRYPTO, 2009.

[i.97] L. Ducas and A. Durmus: "Ring-LWE in polynomial rings" in PKC, 2012.

[i.98] C. Peikert: "How (not) to instantiate Ring-LWE" in SCN, 2016.

[i.99] J. H. Cheon, K. Han, J. Kim, C. Lee and Y. Son: "A practical post-quantum public-key cryptosystem based on spLWE" in ICISC, 2016.

[i.100] L. Ducas, V. Lyubashevsky and T. Prest: "Efficient identity-based encryption over NTRU lattices" in ASIACRYPT, 2014.

[i.101] J. Fan and F. Vercauteren: "Somewhat practical fully homomorphic encryption", IACR ePrint Archive 2012/144, 2012.

[i.102] D. Micciancio and C. Peikert: "Hardness of SIS and LWE with small parameters" in CRYPTO, 2013.

[i.103] J. Buchmann, E. Gopfert, R. Player and T. Wunderer: "On the hardness of LWE with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack" in AFRICACRYPT, 2016.

[i.104] M. Albrecht: "On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL" in EUROCRYPT, 2017.

[i.105] A. Banerjee, C. Peikert and A. Rosen: "Pseudorandom functions and lattices" in EUROCRYPT, 2012.

[i.106] J. H. Cheon, D. Kim, J. Lee and Y. Song: "Lizard: Cut of the tail! Practical post-quantum public-key encryption from LWE and LWR", IACR ePrint Archive 2016/1126, 2016.

[i.107] T. Pöppelmann and T. Güneysu: "Towards practical lattice-based public-key encryption on reconfigurable hardware" in SAC, 2013.

[i.108] L. Tolhuizen, R. Rietman and O. Garcia-Morchon: "Improved key-reconciliations method" IACR ePrint Archive 2017/295, 2017.

[i.109] Z. Brakerski, A. Langlois, C. Peikert, O. Regev and D. Stehlé: "Classical hardness of learning with errors" in STOC, 2013.

[i.110] S. Chatterjee, N. Koblitz, A. Menezes and P. Sarkar: "Another look at tightness II: Practical issues in cryptography", IACR ePrint Archive 2016/360, 2016.

[i.111] K. Eisenträger, S. Hallgren and K. E. Lauter: "Weak instances of PLWE" in SAC, 2014.

[i.112] Y. Elias, K. E. Lauter, E. Ozman and K. E. Stange: "Provably weak instances of Ring-LWE" in CRYPTO, 2015.

[i.113] W. Castryck, I. Iliashenko and F. Vercauteren: "Provably weak instances of Ring-LWE revisited" in EUROCRYPT, 2016.

[i.114] H. Chen, K. E. Lauter and K. E. Stange: "Vulnerable Galois RLWE families and improved attacks", IACR ePrint Archive 2016/193, 2016.

[i.115] R. Lindner and C. Peikert: "Better key sizes (and attacks) for LWE-based encryption" in CT-RSA, 2011.

[i.116] A. Blum, A. Kalai and H. Wasserman: "Noise-tolerant learning, the parity problem, and the statistical query model", Journal of the ACM, vol. 50, no. 4, pp. 506-519, 2003.

[i.117] M. Abe, R. Gennaro, K. Kurosawa and V. Shoup: "Tag-KEM/DEM: A new framework for hybrid encryption and a new analysis of Kurosawa-Desmedt KEM" in EUROCRYPT, 2005.

[i.118] J. Silverman, The arithmetic of elliptic curves, New York: Springer-Verlag, 1992.

[i.119] R. Bröker: "Constructing supersingular curves", J. Comb. Number Theory, vol. 1, no. 3, pp. 269-273, 2009.

[i.120] R. Azarderakhsh, D. Jao, K. Kalach, B. Koziel and C. Leonardi: "Key compression for isogeny-based cryptosystems" in AsiaPKC, 2016.

[i.121] A. Childs, D. Jao and V. Soukharev: "Constructing elliptic curve isogenies in quantum subexponential time", Journal of Mathematical Cryptology, vol. 8, no. 1, pp. 1-29, 2014.

[i.122] J.-F. Biasse, D. Jao and A. Sankar: "A quantum algorithm for computing isogenies between supersingular elliptic curves" in INDOCRYPT, 2014.

[i.123] C. Delfs and S. D. Galbraith: "Computing isogenies between supersingular elliptic curves over Fp" in Designs, Codes and Cryptography, 2014.

[i.124] C. Petit: "Faster algorithms for isogeny problems using torsion point images", IACR ePrint Archive 2017/571, 2017.

[i.125] D. Stebila and M. Mosca: "Post-quantum key exchange for the internet and the Open Quantum-Safe project", IACR ePrint Archive 2016/1017, 2016.

[i.126] P. Longa and M. Naehrig: "Speeding up the number theoretic transform for faster ideal lattice-based cryptography" in CANS, 2016.

3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AES-NI	AES New Instructions
AMD	Advanced Micro Devices
ARM	Advanced RISC machine
AVX	Advanced Vector Extensions
BKZ	Block Korkine-Zolotarev algorithm
DH	Diffie-Hellman
DTLS	Datagram Transport Layer Security
ECDH	Elliptic Curve Diffie-Hellman
GF	Galois Field
IKE	Internet Key Exchange
IoT	Internet of Things
KDF	Key Derivation Function
KEM	Key Encapsulation Mechanism
LWE	Learning With Errors
LWR	Learning With Rounding
NTT	Number-Theoretic Transform
QKD	Quantum Key Distribution
QSC	Quantum-Safe Cryptography
RSA	Rivest-Shamir-Adleman protocol
SIDH	Supersingular Isogeny Diffie-Hellman
SSH	Secure Shell
TLS	Transport Layer Security
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network

4 Quantum-safe key exchanges

4.1 Introduction

Key establishment is a public-key cryptographic primitive which allows two parties to set up a shared secret key. In a key exchange, the shared secret key is securely derived from information contributed by both parties; for example, by exchanging public keys with each other. The standard examples of a key exchange are Diffie-Hellman (DH) and Elliptic Curve Diffie-Hellman (ECDH). In a key transport mechanism one party generates the secret key and securely shares it with the second party; for example, by sending it encrypted under the second party's public key. The standard example of a key transport mechanism is RSA encryption.

DH, ECDH and RSA will all be made insecure by the development of large-scale fault-tolerant quantum computing. The present document discusses proposals in the academic literature for quantum-safe key exchange primitives that could be used directly to replace DH and ECDH. The present document also includes examples of key exchanges constructed from key transport mechanisms.

NOTE 1: Quantum-safe key transport mechanisms that could be used to replace RSA will be considered in a separate document.

NOTE 2: The present document only considers algorithmic key exchange mechanisms. Other mechanisms, based on Quantum Key Distribution (QKD), are considered in ETSI QKD GS 002 [i.1].

Key exchanges can either use short-term ephemeral keys or long-term static keys.

- In ephemeral key exchanges both parties generate new short-term public keys that are only used in a single exchange.

- In static key exchanges one party generates a new short-term public key that is only used in a single exchange, but the other party has a long-term public key that can be used in many exchanges.

Ephemeral key exchanges require on-line interaction between the two parties and are often used to provide perfect forward secrecy. Static key exchanges are more appropriate when on-line interaction is not possible.

The present document considers some representative examples of quantum-safe key exchanges, but does not attempt to be exhaustive. More discussion of quantum-safe primitives can be found in ETSI GR QSC 001 [i.2].

4.2 Use cases

4.2.1 General comments

Most commercial systems will need a general-purpose quantum-safe key exchange that has good security and efficiency properties, and which can be integrated into existing protocols with minimal changes. IoT networks will have more severe resource constraints, but for use cases that involve low-value ephemeral data a lower security solution might be acceptable.

4.2.2 Network security

Network security protocols such as Transport Layer Security (TLS), Datagram TLS (DTLS), Internet Key Exchange (IKE) and Secure Shell (SSH) are ubiquitous. They are used to protect a range of different internet applications including web browsing, instant messaging, Voice over Internet Protocols (VoIP), Virtual Private Networks (VPNs) and remote access. These protocols typically provide confidentiality by performing a key exchange during an initial handshake or setup phase. Many modern protocols such as TLS 1.3 [i.3] or IKEv2 [i.4] also require ephemeral key exchanges for perfect forward security.

The performance of the quantum-safe key exchange is an important consideration. Protocol limitations or practical issues such as packet fragmentation also mean that it can be more difficult to use key exchanges that have large public keys. More discussion of the practical issues around the integration and deployment of quantum-safe cryptography into real-world systems can be found in ETSI GR QSC 003 [i.5].

4.2.3 Internet of Things

The Internet of Things (IoT) is a term which covers a wide range of different locally-networked or internet-connected objects. Applications include smart home devices, wearable technology, intelligent transport systems, sensor networks and industrial control systems. A common characteristic of IoT devices is that they have limited power, memory, bandwidth, or computational resources. This means that the security of IoT networks has often relied on pre-shared symmetric keys, but public-key primitives such as ECDH are increasingly being used where resource constraints allow this.

Quantum-safe key exchanges that have small public keys will be needed for IoT applications with limited bandwidth. Similarly, devices with limited power or computational resources will require key exchanges that have efficient key generation. However, it is likely that constrained environments will not be able to support perfect forward secrecy or algorithm negotiation.

4.3 Candidate primitives

The only key exchanges described in ETSI GR QSC 001 [i.2] are either based on lattices or on supersingular isogenies. The present document therefore considers the following key exchanges:

- Learning with Errors (LWE) key exchange (clause 6.2);
- Ring-LWE key exchange (clause 6.3); and
- Supersingular Isogeny Diffie-Hellman (SIDH) key exchange (clause 7).

Although not widely used, it is also possible to construct a key exchange from a key transport mechanism. The key transport mechanisms described in ETSI GR QSC 001 [i.2] are based on lattices, codes and multivariate systems.

Consequently, the present document also includes examples of key exchanges based on the following key transport mechanisms:

- Niederreiter key transport (clause 8.2); and
- NTRU key transport (clause 8.3).

5 Implementation considerations

5.1 Introduction

The two most important considerations for deploying quantum-safe key exchanges are the size of the parameters and the performance of the algorithms at an appropriate security level. However, a scheme or implementation that is secure when used for purely ephemeral key exchanges can be insecure when used for static exchanges due to active attacks or side-channel vulnerabilities.

5.2 Active security

5.2.1 Invalid key attacks

Key exchanges can be vulnerable to active attacks where one party uses an invalid public key [i.6], or provokes a failure in the protocol, to learn information about the other party's private key. This is not usually a concern in an ephemeral key exchange since the private key will only be used for that particular exchange and a new one will be generated for the next exchange. However, if one of the public keys is static then it is possible to use these techniques to recover the full static private key over a number of key exchanges. One solution to this problem is to modify the key exchange protocol so that it includes a key validation step.

5.2.2 Key validation

In DH and ECDH the public key can be directly validated by checking that it belongs to the correct subgroup. For some quantum-safe primitives this is not possible as valid public keys are indistinguishable from invalid ones. Instead, it is necessary to validate keys indirectly using techniques such as the Fujisaki-Okamoto transform [i.7]. The responder sends the initiator their ephemeral private key encrypted under the shared secret key that the responder derived from the key exchange. The initiator can then attempt to decrypt the responder's ephemeral private key and use it to reconstruct the responder's side of the exchange. If the reconstructed exchange does not match the original exchange exactly then the initiator knows that either the original key exchange failed or the responder did not perform their side of the exchange honestly.

NOTE 1: After a key exchange involving the Fujisaki-Okamoto transform the responder will have revealed their private key to the initiator. This means that it is important for the responder to generate a new private key for each key exchange.

NOTE 2: The original Fujisaki-Okamoto transform [i.7] has a proof of security against classical adversaries, but this does not hold for quantum adversaries. It is necessary to modify the transform slightly [i.8] to provide a security proof in this case.

5.2.3 Performance impact

If the ephemeral private keys are deterministically generated from a seed then the communication overhead from using the Fujisaki-Okamoto transform is small since it only adds an encryption of the seed to the information that the responder sends to the initiator. Similarly, the Fujisaki-Okamoto transform does not significantly increase the cost of computing the responder's side of exchange since the only addition is the encryption of the seed. However, the initiator recreates the responder's side of the exchange which almost doubles the total computational cost for the initiator.

NOTE: Ephemeral keys are only intended to be used once, but they are often cached and used in a number of key exchanges over a relatively short time period to improve performance on servers with high loads. Cached ephemeral keys can be vulnerable to the same active attacks as static keys. In this case the performance overhead from using the Fujisaki-Okamoto transform to mitigate these attacks is more than the cost of generating a fresh ephemeral key for each exchange.

5.3 Side-channel protection

5.3.1 Side-channel vulnerabilities

Side-channel attacks use secret information obtained from the physical implementation of a cryptosystem to recover the private key rather than attacking the underlying algorithms. These attacks can either be passive or active. Passive attacks are performed by observing and analysing physical quantities, such as execution time [i.9], power consumption [i.10] or electromagnetic emission. In active attacks, the adversary manipulates the device by modifying its inputs, its environment or both, with the goal being to induce abnormal behaviour in the device and exploiting it to perform the attack. Although many attacks require close physical proximity or access to a device, it has been shown that timing attacks can also be carried out remotely across a network [i.11]. Security against these various forms of side-channel attack is therefore a vital consideration in the design and implementation of a physical cryptosystem.

5.3.2 Side-channel mitigations

Timing attacks include a range of techniques from simple power analysis to cache attacks, which exploit differences in execution time due to branching based on private data in order to extract some information. These attacks can generally be blocked by ensuring a constant execution time. However, it can be challenging for certain target platforms [i.12], or can be difficult to achieve due to compiler optimizations. Power analysis countermeasures aim to make the power consumption independent from the secret data, using either hiding or masking techniques [i.13]. Hiding looks to shuffle the order of operations, or increasing the amount of noise so that an adversary does not know where the private key dependent leakage occurs. Masking looks to break the link between the secret value being operated on and the physical power consumption through the addition of randomness [i.14].

5.3.3 Performance impact

There is a trade-off associated to side-channel countermeasures in terms of achieved security level versus performance, with the implementation of countermeasures potentially costly in terms speed or memory. The performance cost of constant time operations varies depending on the underlying algorithm, but implementation techniques mean it does not need to be prohibitive. Similarly, the addition of noise can be relatively low-cost in terms of performance, although it requires additional energy which might be an issue for embedded devices and is not secure against attacks. On the other hand, masking is provably secure [i.15] but can incur performance penalties of an order of magnitude or more.

6 Learning with Errors

6.1 Introduction

Cryptography based on the hardness of lattice problems [i.16] and [i.17] has been around for over twenty years and there have been schemes using structured lattices [i.18] for almost as long. Much of modern lattice-based cryptography, and most modern lattice-based encryption schemes, relies on the security of the LWE problem [i.19] or its structured lattice version, Ring-LWE [i.20] and [i.21]. Although the first LWE and Ring-LWE schemes can clearly be viewed as analogous to ElGamal encryption and the possibility of a Diffie-Hellman style key exchange was mentioned in [i.22], concrete proposals for key exchanges are more recent [i.23] and [i.24].

Clause 6.2 gives a description of a LWE key exchange and clause 6.3 gives a description of a Ring-LWE key exchange. These are based on the general scheme presented in [i.24] and variations of the basic key exchanges are described in clause A.1. Clause 6.4 discusses some implementation considerations, clause 6.5 lists proposed parameters, and clause 6.6 gives performance estimates. Further discussion of security considerations can be found in clause A.2. Performance comparisons with other key exchanges can be found in annex C.

6.2 LWE key exchange

6.2.1 Overview

The LWE key exchange follows the same general format as a Diffie-Hellman key exchange: each party generates a key pair and sends their public key to the other party. However, as small noise terms are added to the public keys, the "shared values" that each party computes will differ by a small amount. Consequently, one party sends the other party an additional check field so that they are both able to compute the same secret key with reasonable probability. This means that the LWE key exchange is not symmetric as the two parties involved behave differently. It will be clearer to describe the key exchange in terms of an initiator, who starts the exchange, and a responder.

The LWE key exchange uses a public matrix A. This can be a fixed system parameter chosen by a trusted party during the setup of the scheme as in [i.24], freshly generated by the initiator for each key exchange as in [i.25], or agreed upon in some other manner providing a trade-off between performance and security as in [i.26]. Further discussion can be found in clause A.1.3.

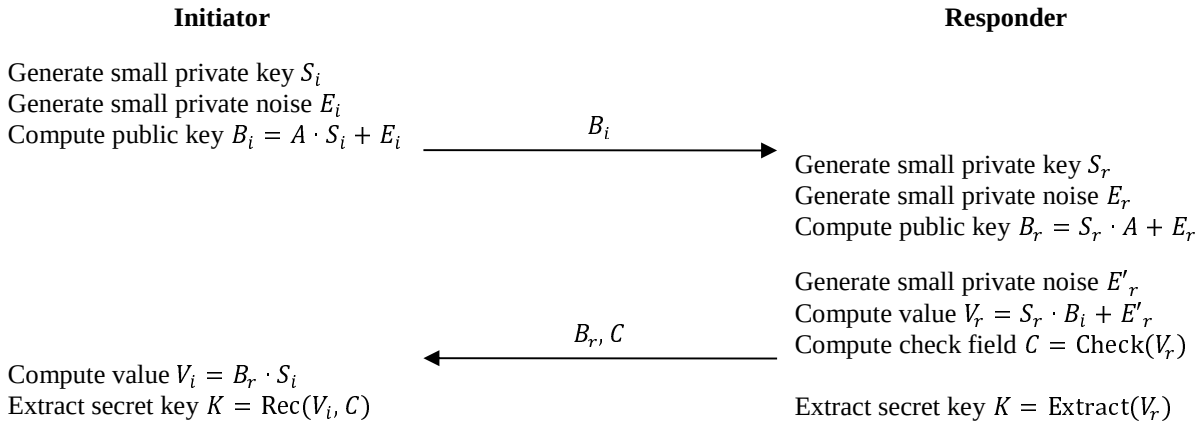

Figure 1: LWE key exchange with public matrix A

The LWE key exchange proceeds as follows and is illustrated in figure 1:

1) The initiator generates a private key S_i and private noise term E_i, both of which are matrices with small entries. The initiator's corresponding public key $B_i = A \cdot S_i + E_i$ is computed from their private key S_i, private noise term E_i and the public matrix A. The initiator sends their public key B_i to the responder.

2) The responder generates a private key S_r and private noise term E_r, both of which are matrices with small entries. The responder's corresponding public key $B_r = S_r \cdot A + E_r$ is computed from their private key S_r, private noise term E_r and the public matrix A.

3) The responder generates a second noise term E'_r, which is a matrix with small entries, and forms the intermediate value $V_r = S_r \cdot B_i + E'_r$ from their private key S_r, the initiator's public key B_i, and their second noise term E'_r. The responder computes a check field C from V_r and sends their public key B_r and check field C to the initiator.

4) The responder extracts a shared secret key K_r from V_r.

5) The initiator forms the intermediate value $V_i = B_r \cdot S_i$ from their private key S_i and the responder's public key B_r. The initiator uses the check field C to extract a shared secret key K_i from V_i. The shared secret keys K_i and K_r will be the same with reasonable probability.

6.2.2 Public parameters

The public parameters for the LWE key exchange are:

- A pair of integers (n, m) which give the dimensions of the matrices;
- An integer q which gives the modulus for the matrix entries;
- A probability distribution χ over $\mathbb{Z}/q\mathbb{Z}$ which is used to sample small matrix entries;
- An integer B giving the number of bits to use in key extraction; and
- A public $n \times n$ matrix A with entries chosen independently and uniformly at random from $\mathbb{Z}/q\mathbb{Z}$.

The LWE modulus q is typically either prime or a power of two. The description below will follow [i.25] and assume that q is a power of two to simplify key extraction and reconciliation.

The probability distribution will be a discrete Gaussian over $\mathbb{Z}$ with small standard deviation σ, which is then reduced modulo q.

The public matrix A can either be a fixed public parameter or can vary and forms part of the initiator's public key. If it varies then the initiator needs to send the responder A or a seed which can be used to generate A deterministically.

NOTE: The one-dimensional discrete Gaussian can be replaced with a simpler distribution to improve efficiency. See clause A.1.2 for more details.

6.2.3 Key generation

Key generation in the LWE key exchange is different for the initiator and for the responder.

The initiator's private key S_i is an $n \times m$ matrix generated by independently sampling the entries from χ. The initiator's public key B_i is the $n \times m$ matrix computed by first generating an $n \times m$ noise matrix E_i, with entries independently sampled from χ, and then forming $B_i = A \cdot S_i + E_i$.

The responder's private key S_r is an $m \times n$ matrix generated by independently sampling the entries from χ. The responder's public key B_r is the $m \times n$ matrix computed by first generating an $m \times n$ noise matrix E_r, with entries independently sampled from χ, and then forming $B_r = S_r \cdot A + E_r$.

6.2.4 Key extraction

Given the initiator's public key B_i, the responder's intermediate value V_r is the $m \times m$ matrix computed by first generating an $m \times m$ noise matrix E'_r, with entries independently sampled from χ, and then forming $V_r = S_r \cdot B_i + E'_r$. The shared secret key K_r is extracted from V_r by rounding the B most significant bits from each entry; i.e., the (i, j)-th entry of the shared secret key is:

$$K_r[i,j] = \mathrm{Round}\left(\frac{2^B}{q} V_r[i,j]\right) \bmod 2^B$$

where $V_r[i,j]$ is the (i,j)-th entry of the responder's intermediate value viewed as an integer in $[-q/2, q/2-1]$. As V_r is indistinguishable from a random $m \times m$ matrix over $\mathbb{Z}/q\mathbb{Z}$ and q is assumed to be a power of two, this produces an unbiased secret key.

6.2.5 Reconciliation

Given the responder's public key B_r, the initiator's intermediate value is the $m \times m$ matrix $V_i = B_r \cdot S_i$. This differs from the responder's intermediate value by:

$$V_r - V_i = S_r \cdot E_i - E_r \cdot S_i + E'_r.$$

In particular, rounding the B most significant bits from each entry of V_i might not recover the same secret key K_r. The probability of recovering K_r is improved by using a reconciliation mechanism: the responder sends the initiator a check field which allows the initiator to correct errors in their key extraction.

More specifically, the responder's check field C is the $m \times m$ matrix over $\mathbb{Z}/2\mathbb{Z}$ obtained by cross-rounding the entries of V_r; i.e., the (i,j)-th entry of the check field is:

$$C[i,j] = \mathrm{Floor}\left(\frac{2^{B+1}}{q} V_r[i,j]\right) \bmod 2$$

where $V_r[i,j]$ is the (i,j)-th entry of the responder's intermediate value viewed as an integer in $[-q/2, q/2-1]$.

The initiator uses the responder's check field C to adjust the rounding when extracting the shared secret key K_i from their intermediate value V_i. The (i,j)-th entry of the shared secret key obtained from reconciliation is:

$$K_i[i,j] = \mathrm{Round}\left(\frac{2^B}{q} V_i[i,j] + \frac{1}{4}(2C[i,j]-1)\right) \bmod 2^B$$

where $V_i[i,j]$ is the (i,j)-th entry of the initiator's intermediate value viewed as an integer in $[-q/2, q/2-1]$.

NOTE: Reconciliation improves the probability that the initiator and responder both extract the same shared secret key, but key exchange failures might still occur. See clause A.1.4 for more details.

6.3 Ring-LWE key exchange

6.3.1 Overview

The Ring-LWE key exchange follows a similar format to the LWE exchange with matrices replaced by polynomials. This reduces the size of the public keys and allows more efficient arithmetic by introducing structure to the lattices.

Each party generates a key pair and sends their public key to the other party, but as small noise terms are added to the public keys the "shared values" that each party computes will differ by a small amount. The responder therefore also sends the initiator an additional check field so that they are both able to compute the same shared secret key with reasonable probability.

The Ring-LWE key exchange uses a public polynomial A. This can be a fixed system parameter chosen by a trusted party during the setup of the scheme as in [i.24], freshly generated by the initiator for each key exchange as in [i.27], or agreed upon in some other manner providing a trade-off between performance and security. Further discussion can be found in clause A.1.3.

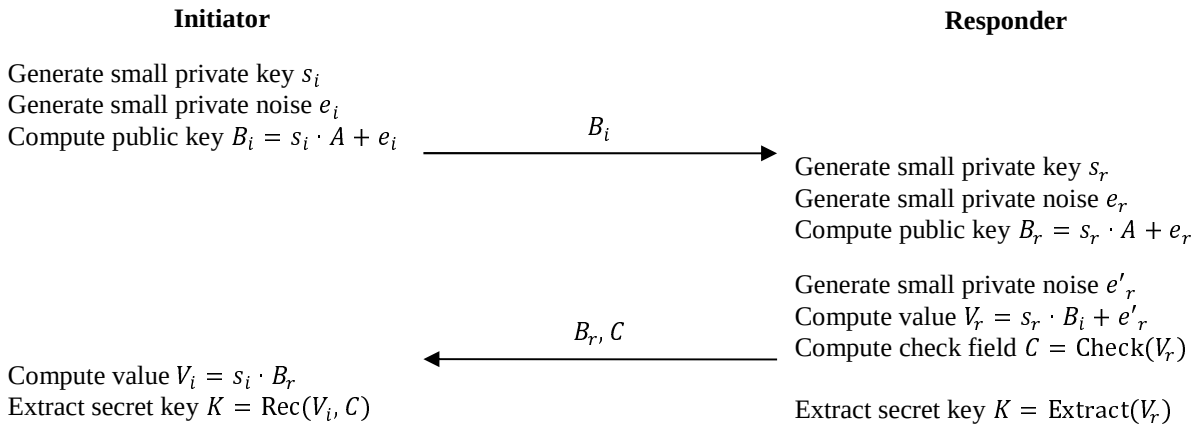

Figure 2: Ring-LWE key exchange with public polynomial A

The Ring-LWE key exchange proceeds as follows and is illustrated in figure 2:

1) The initiator generates a private key s_i and private noise term e_i, both of which are polynomials with small coefficients. The initiator's corresponding public key $B_i = s_i \cdot A + e_i$ is computed from their private key s_i, private noise term e_i and the public polynomial A. The initiator sends their public key B_i to the responder.

2) The responder generates a private key s_r and private noise term e_r, both of which are polynomials with small coefficients. The responder's corresponding public key $B_r = s_r \cdot A + e_r$ is computed from their private key s_r, private noise term e_r and the public polynomial A.

3) The responder generates a second noise term e'_r, which is a polynomial with small coefficients, and forms the intermediate value $V_r = s_r \cdot B_i + e'_r$ from their private key s_r, the initiator's public key B_i, and their second noise term e'_r. The responder computes a check field C from V_r and sends their public key B_r and check field C to the initiator.

4) The responder extracts a shared secret key K_r from V_r.

5) The initiator forms the intermediate value $V_i = s_i \cdot B_r$ from their private key S_i and the responder's public key B_r. The initiator uses the check field C to extract a shared secret key K_i from V_i. The shared secret keys K_i and K_r will be the same with reasonable probability.

6.3.2 Public parameters

The public parameters for the Ring-LWE key exchange are:

- An integer m which defines the ring $R = \mathbb{Z}[x]/(\Phi_m(x))$ where $\Phi_m(x)$ is the m-th cyclotomic polynomial;
- A prime integer q which gives the modulus for the ring $R_q = R/qR$;
- A probability distribution χ which is used to sample small polynomials from R; and
- A public polynomial $A \in R_q$ with coefficients chosen independently and uniformly at random from $\mathbb{Z}/q\mathbb{Z}$.

The integer m is typically a prime or a power of two and the prime q is chosen to be congruent to 1 modulo m. The description below follows [i.28] and [i.29] and assumes that m is a power of two. In this case the cyclotomic polynomial is $\Phi_m(x) = 1 + x^{m/2}$. Let $n = m/2$ denote the degree of the polynomial $\Phi_m(x)$ and so the dimension of the ring R.

The probability distribution χ used to sample small polynomials from R is usually chosen to be a discrete Gaussian. In the power of two case, this can be achieved by generating polynomials of the form $e = e_0 + e_1 x + \cdots + e_{n-1} x^{n-1} \in R$ where the coefficients e_i are sampled independently from a one-dimensional discrete Gaussian distribution of small standard deviation σ.

The public polynomial A can either be a fixed public parameter or can vary and forms part of the initiator's public key. If it varies then the initiator needs to send the responder A or a seed which can be used to generate A deterministically.

NOTE 1: Power-of-two cyclotomic rings offer more efficient polynomial arithmetic than prime cyclotomic rings, but are less flexible for parameter selection. See clause A.1.1 for more details.

NOTE 2: The one-dimensional discrete Gaussian can be replaced by a simpler distribution to improve efficiency. See clause A.1.2 for more details.

6.3.3 Key generation

Key generation for the Ring-LWE key exchange is the same for the initiator and for the responder.

The private key s is a polynomial in R sampled from the distribution χ. The corresponding public key B is the polynomial in R_q computed by first generating a noise term $e \in R$ sampled from χ and then forming:

$$B = s \cdot A + e$$

where s and e are viewed as elements of R_q by reducing their coefficients modulo q.

6.3.4 Key extraction

Given the initiator's public key B_i, the responder's intermediate value V_r is the polynomial in R_q computed by first generating a noise term $e'_r \in R$ sampled from χ and then forming $V_r = s_r \cdot B_i + e'_r$ where s_r and e'_r are viewed as elements of R_q by reducing their coefficients modulo q. The shared secret key is extracted from V_r by randomized rounding of the most significant bit from each coefficient. The i-th entry of the shared secret key is:

$$K_r[i] = \text{Round}\left(\frac{2}{q} V_r[i] + \frac{1}{2q} e''_r[i]\right) \bmod 2$$

where $V_r[i]$ is the i-th coefficient of the responder's intermediate value viewed as an integer in $[-(q+1)/2, (q-1)/2]$ and $e''_r[i]$ is sampled from $\{-1,0,1\}$ with probabilities $p_{-1} = 1/4$, $p_0 = 1/2$ and $p_1 = 1/4$.

As q is odd, the additional randomization from e''_r is needed to avoid rounding biases in the shared secret key.

6.3.5 Reconciliation

Given the responder's public key B_r, the initiator's intermediate value is the polynomial $V_i = s_i \cdot B_r$ in R_q. This differs from the responder's intermediate value by:

$$V_r - V_i = s_r \cdot e_i - s_i \cdot e_r + e'_r.$$

In particular, rounding the most significant bit from each coefficient of v_i might not recover the same secret key K_r. The probability of recovering K_r is improved by using a reconciliation mechanism: the responder sends the initiator a check field which allows the initiator to correct errors in their key extraction.

More specifically, the responder's check field C is the element in $\{0,1\}^n$ obtained by cross-rounding the coefficients of V_r; i.e., the i-th entry of the check field is:

$$C[i] = \text{Floor}\left(\frac{4}{q} V_r[i] + \frac{1}{q} e''_r[i]\right) \bmod 2$$

where $V_r[i]$ is the i-th coefficient of the responder's intermediate value viewed as an integer in $[-(q+1)/2, (q-1)/2]$ and e''_r is the additional noise term that was used by the responder in key extraction.

The initiator uses the responder's check field C to adjust the rounding when extracting the shared secret key K_i from their intermediate value V_i. The i-th entry of the shared secret key obtained from reconciliation is:

$$K_i[i] = \text{Round}\left(\frac{2}{q}V_i[i] + \frac{1}{4}(2C[i]-1)\right) \bmod 2$$

where $V_i[i]$ is the i-th coefficient of the initiator's intermediate value viewed as an integer in $[-(q+1)/2, (q-1)/2]$. In this case randomized rounding is not necessary as the aim is to recover the responder's shared secret key.

NOTE: Reconciliation improves the probability that the initiator and responder both extract the same shared secret key, but key exchange failures might still occur. See clause A.1.4 for more details.

6.4 Implementation considerations

6.4.1 Active security

Fluhrer [i.30] shows that if the initiator and responder fail to extract the same shared secret key following the Ring-LWE key exchange then the responder can learn some information about the initiator's private key. Further, Ding et al [i.31] describe an active attack where a dishonest responder using malformed public keys can recover the responder's private key with only $2q$ key exchange attempts. It is not possible to validate Ring-LWE public keys directly as malformed keys are indistinguishable from correctly formed ones. Instead, Peikert's actively secure key exchange [i.24] protects against dishonest responders by using the Fujisaki-Okamoto transform. See clause A.2.3 for more details.

The Fujisaki-Okamoto transform is not enough to block active attacks against static key exchanges by itself. If the failure probability is high enough, then the responder can still try to recover the initiator's static private key using honest key exchange failures. The total cost of the attack will have increased significantly as the responder now needs many key generation attempts before they find a valid ephemeral key pair likely to cause a key exchange failure. However, this key generation can be performed off-line and the number of on-line exchange attempts with the initiator will be similar to the original attack. Consequently, for static key exchanges it is necessary to choose parameters which have a negligible probability of failure.

6.4.2 Side-channel protection

An introduction to side-channel protection for lattice-based cryptography is provided in [i.32]. LWE and Ring-LWE cryptosystems are vulnerable to both timing and power analysis attacks [i.33], [i.34] and [i.35]. Lattice-based cryptosystems often use optimizations to accelerate performance, such as early exits or the use of pre-computed tables to minimize latency, which have significant impact on the timings of a design. Discrete Gaussian sampling, an operation often used in LWE and Ring-LWE schemes, is also vulnerable to timing attacks, due to inherent non-constant run times or the use of cache access patterns [i.33]. Countermeasures for side-channel protection, such as constant time implementations, masking and random shuffling, have already been used to protect lattice-based cryptosystems [i.36], [i.37] and [i.38]. However, further investigation is necessary to ensure there is no information leakage after implementation.

6.5 Parameter selection

6.5.1 LWE proposed parameters

Table 1 lists parameters that have been proposed for the LWE key exchange together with the corresponding public key lengths, claimed security levels and estimated key exchange failure probabilities.

Table 1: Proposed parameter sets for the LWE key exchange

Scheme	n	m	q	B	σ	Public key length	Security level:		Failure probability	Notes
							Classical	Quantum		
Frodo	752	8	2^{15}	4	1,32	11 280 bytes	144	130	2^{-39}	[i.25]
Frodo	864	8	2^{15}	4	1,32	12 960 bytes	177	161	2^{-34}	[i.25]
spKEX	738	8	2^{14}	4	2,309	8 118 bytes	141	128	2^{-42}	[i.26]

6.5.2 Ring-LWE proposed parameters

Table 2 lists parameters that have been proposed for the Ring-LWE key exchange together with the corresponding public key lengths, claimed security levels and estimated key exchange failure probabilities.

Table 2: Proposed parameter sets for the Ring-LWE key exchange

Scheme	n	q	σ	Public key Length	Security level:		Failure probability	Notes
					Classical	Quantum		
BCNS	1 024	$2^{32} - 1$	3,192	4 096 bytes	164	82	$2^{-131072}$	[i.28] and note 1
Singh	1 024	40 961	3,192	2 048 bytes	256	---	2^{-91}	[i.39] and note 2
New Hope	1 024	12 289	2,828	1 792 bytes	281	255	2^{-60}	[i.27] and note 3
Singh-Chopra	540	41 117	3,192	1 080 bytes	165	146	2^{-87}	[i.40]

NOTE 1: The only quantum security estimate given in [i.28] assumes a square root speed up due to Grover's algorithm.
NOTE 2: Reference [i.39] claims that the parameters have at least 256 bits of classical security, but does not give a quantum security estimate.
NOTE 3: The quoted failure probability is achieved using a more complicated reconciliation mechanism than the scheme described in clause 6.3.5.

6.5.3 Security estimates

The claimed security levels given in tables 1 and 2 are taken directly from the referenced documents. In each case, the authors have attempted to estimate the cost of the relevant lattice attacks against the proposed parameters. However, variations in the lattice attacks being considered or differences in the lattice reduction heuristics being used can lead to significantly different security estimates. Clause A.2.2 contains a comparison of the security estimate obtained following the concrete approach in [i.41] and the conservative approach in [i.27].

6.6 Performance

6.6.1 Performance on a 64-bit processor

Table 3 gives performance estimates (in millions of clock cycles) for implementations of the LWE and Ring-LWE key exchanges on a 64-bit Intel® Xeon® E5 (Sandy Bridge) processor or a 64-bit Intel® Core™ i5 (Haswell) processor. The first performance column refers to the initiator's key generation (step 1 in clauses 6.2.1 and 6.3.1); the second column refers to the responder's key generation, check field calculation and key extraction (steps 2-4); and the third column refers to the initiator's reconciliation (step 5).

Table 3: Performance of the LWE and Ring-LWE key exchanges on a 64-bit processor

Scheme	Performance:			Communication:		Notes
	Initiator (1)	Responder (2-4)	Initiator (5)	Initiator	Responder	
Frodo	2,94 M cycles	3,45 M cycles	0,338 M cycles	11 377 bytes	11 296 bytes	[i.25] and note 1
spKEX	2,45 M cycles	2,22 M cycles	0,024 M cycles	8 150 bytes	8 142 bytes	[i.26] and note 2
BCNS	2,48 M cycles	4,00 M cycles	0,482 M cycles	4 096 bytes	4 224 bytes	[i.28]
New Hope	0,258 M cycles	0,385 M cycles	0,086 M cycles	1 824 bytes	2 048 bytes	[i.27] and note 3
New Hope	0,089 M cycles	0,111 M cycles	0,019 M cycles	1 824 bytes	2 048 bytes	[i.27] and note 4

NOTE 1: The performance estimates are for an implementation on a 64-bit Intel® Xeon® E5 (Sandy Bridge) processor. Key generation times for the initiator and responder include the cost of constructing the public matrix from a seed. Reference [i.25] states that this takes approximately 1,4 M cycles.
NOTE 2: The performance estimates are for an implementation on a 64-bit Intel® Core™ i5 (Haswell) processor. Key generations times for the initiator and responder include the cost of permutating the public matrix.
NOTE 3: The performance estimates are for an unoptimized implementation on a 64-bit Intel® Xeon® E5 (Sandy Bridge) processor. Key generation times for the initiator and responder include the cost of constructing the public polynomial from a seed.
NOTE 4: The performance estimates are for an optimized implementation on a 64-bit Intel® Xeon® E5 (Sandy Bridge) processor using AVX2 instructions. Key generation times for the initiator and responder include the cost of constructing the public polynomial from a seed.

6.6.2 Performance on a 32-bit embedded processor

Table 4 gives performance estimates (in millions of clock cycles) for an implementation of the LWE key exchange on a single board computer containing a 32-bit ARM® Cortex®-A8 core.

Table 4: Performance of the LWE key exchange on a 32-bit embedded processor

Scheme	Performance:			Communication:		Notes
	Initiator (1)	Responder (2-4)	Initiator (5)	Initiator	Responder	
Frodo	77,5 M cycles	80,2 M cycles	1,09 M cycles	11 377 bytes	11 296 bytes	[i.25] and note

NOTE: Key generation times for the initiator and responder include the cost of constructing the public matrix from a seed.

6.6.3 Performance on 32-bit microcontrollers

Table 5 gives performance estimates (in clock cycles) for implementations of the Ring-LWE key exchange on a pair of microcontrollers containing either an 32-bit ARM® Cortex®-M4 or an ARM® Cortex®-M0 cores.

Table 5: Performance of the Ring-LWE key exchange on 32-bit microcontrollers

Scheme	Performance:			Communication:		Notes
	Initiator (1)	Responder (2-4)	Initiator (5)	Initiator	Responder	
New Hope	0,964 M cycles	1,42 M cycles	0,179 M cycles	1 824 bytes	2 048 bytes	[i.29] and note 1
New Hope	1,17 M cycles	1,74 M cycles	0,299 M cycles	1 824 bytes	2 048 bytes	[i.29] and note 2

NOTE 1: The performance estimates are for New Hope on a 32-bit ARM® Cortex®-M4 core. Key generation times for the initiator and responder include the cost of constructing the public polynomial from a seed. Reference [i.29] states that this takes 0,294 M cycles. The implementation used 22,8 Kbytes of internal flash memory.
NOTE 2: The performance estimates are for New Hope on a 32-bit ARM® Cortex®-M0 core. Key generation times for the initiator and responder include the cost of constructing the public polynomial from a seed. Reference [i.29] states that this takes 0,381 M cycles. The implementation used 30,2 Kbytes of internal flash memory.

6.7 Summary

The LWE and Ring-LWE problems are gaining popularity as techniques for constructing quantum-safe cryptosystems and there have been a large number of concrete proposals for both key exchanges and other public-key primitives.

Ring-LWE key exchanges have reasonably compact public keys and good performance even on constrained devices so they would be suitable as general-purpose drop-in replacements for ephemeral DH and ECDH key exchanges. If they are used to replace static DH or ECDH key exchanges then they can be vulnerable to attacks that exploit invalid public keys or key exchange failures unless appropriate countermeasures are taken.

LWE key exchanges offer more flexibility in parameter selection by avoiding the use of structured lattices, but are not as efficient so could only replace DH and ECDH key exchanges in applications that can tolerate larger public keys. The recently proposed Crystals suite [i.42] and [i.43] attempts to balance flexibility and efficiency by using Module-LWE [i.44].

The provable security results give a level of confidence in the proposals, the practical security of the underlying lattice problems is increasingly well understood, and there is on-going research into side-channel attacks and mitigation.

7 Supersingular isogenies

7.1 Introduction

Cryptography based on isogenies between supersingular elliptic curves is a relatively new area of research. The SIDH key exchange was first introduced by Jao and De Feo [i.45] in 2011. It has subsequently been refined in follow-up work by De Feo, Jao and Plût [i.46] and Costello, Longa and Naehrig [i.47].

Clause 7.2 gives a high-level description of the SIDH key exchange and a brief outline of the mathematical background can be found in clause B.1. Clause 7.3 discusses some implementation considerations, clause 7.4 lists proposed parameters, and clause 7.5 gives performance estimates. Further discussion of security considerations can be found in clause B.2. Performance comparisons with other key exchanges can be found in annex C.

7.2 SIDH key exchange

7.2.1 Overview

The SIDH key exchange follows the same general format as a Diffie-Hellman key exchange: each party generates a key pair and sends their public key to the other party. However, the two parties use different subgroups of the elliptic curve to construct their key pairs. This means that the key exchange is not symmetric and so it will be clearer to describe the key exchange in terms of an initiator, who starts the exchange, and a responder.

The SIDH key exchange uses a public supersingular curve E with two pairs of auxiliary points $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$. These can either involve a specified curve as in [i.47] or a curve generated randomly by a trusted party during the setup of a scheme as in [i.46]. Further discussion of parameter generation can be found in clause B.1.2.

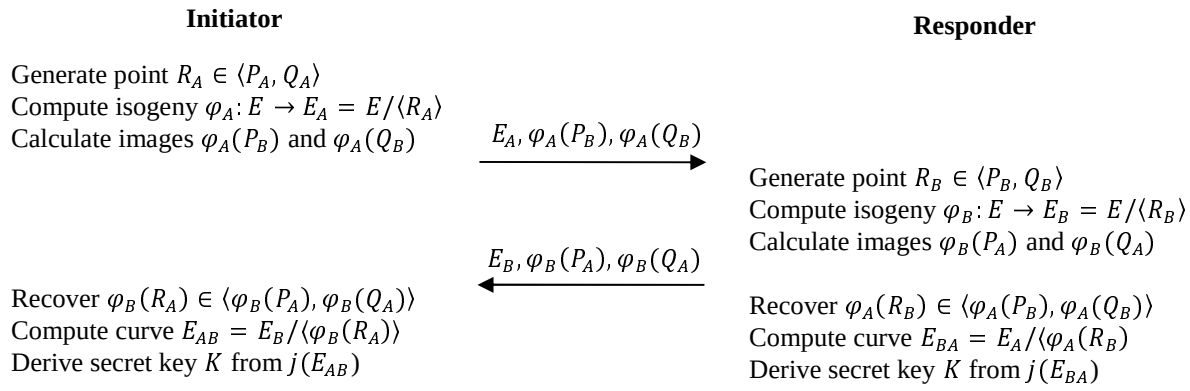

Figure 3: SIDH key exchange

The SIDH key exchange proceeds as follows and is illustrated in figure 3:

1) The initiator generates a point $R_A \in \langle P_A, Q_A \rangle$, uses this to compute a private isogeny $\varphi_A: E \to E_A = E/\langle R_A \rangle$ and then calculates the images $\varphi_A(P_B)$ and $\varphi_A(Q_B)$. The initiator sends the curve E_A and points $\varphi_A(P_B)$, $\varphi_A(Q_B)$ to the responder.

2) The responder generates a point $R_B \in \langle P_B, Q_B \rangle$, uses this to compute a private isogeny $\varphi_B: E \to E_B = E/\langle R_B \rangle$ and then calculates the images $\varphi_B(P_A)$ and $\varphi_B(Q_A)$. The responder sends the curve E_B and points $\varphi_B(P_A)$, $\varphi_B(Q_A)$ to the initiator.

3) The responder uses the initiator's public points $\varphi_A(P_B)$, $\varphi_A(Q_B)$ to recover the image $\varphi_A(R_B)$ of R_B under the initiator's private isogeny φ_A. The responder computes the curve $E_{BA} = E_A/\langle \varphi_A(R_B) \rangle$ and derives the shared secret key from the j-invariant of E_{BA}.

4) The initiator uses the responder's public points $\varphi_B(P_A)$ and $\varphi_B(Q_A)$ to recover the image $\varphi_B(R_A)$ of R_A under the responder's private isogeny φ_B. The initiator computes the curve $E_{AB} = E_B/\langle \varphi_B(R_A) \rangle$ and derives the shared secret key from the j-invariant of E_{AB}.

7.2.2 Public parameters

The public parameters for the SIDH key exchange are:

- A prime p of the form $p = l_A^{e_A} l_B^{e_B} f \pm 1$ where l_A and l_B are small primes and f is an optional cofactor;
- A fixed supersingular elliptic curve E over $\mathrm{GF}(p^2)$;
- A pair of auxiliary points $\{P_A, Q_A\}$ that generate the $l_A^{e_A}$-torsion subgroup $E[l_A^{e_A}]$; and
- A pair of auxiliary points $\{P_B, Q_B\}$ that generate the $l_B^{e_B}$-torsion subgroup $E[l_B^{e_B}]$.

The prime is typically chosen to have the form $p = l_A^{e_A} l_B^{e_B} f - 1$ to allow efficient arithmetic in $\mathrm{GF}(p^2)$. The small primes are typically chosen to be $l_A = 2$ and $l_B = 3$ as these allow efficient isogeny calculations. The exponents e_A and e_B are chosen so that $l_A^{e_A}$ and $l_B^{e_B}$ are approximately the same size to balance the sizes of the torsion subgroups for the initiator and responder.

When $p \equiv 3 \pmod 4$ the supersingular elliptic curve E can be chosen to be $y^2 = x^3 + x$. However, it is also possible to generate a random supersingular curve over $\mathrm{GF}(p^2)$ during the setup of the scheme. The process for generating random parameters is outlined in clause B.1.2.

7.2.3 Key generation

The private key consists of a pair of integers and the public key consists of an elliptic curve over $\mathrm{GF}(p^2)$ isogenous to E and a pair of points on the curve. However, the key generation process is slightly different for the initiator and responder.

The initiator randomly chooses a pair of integers $m_A, n_A \in \mathbb{Z}/l_A^{e_A}\mathbb{Z}$ and forms the point $R_A = m_A P_A + n_A Q_A \in E\left[l_A^{e_A}\right]$. The initiator then computes the isogenous curve E_A corresponding to the isogeny $\varphi_A: E \longrightarrow E_A = E/\langle R_A \rangle$ given by the kernel $\langle R_A \rangle$. Finally, the initiator computes the images $\varphi_A(P_B)$ and $\varphi_A(Q_B)$ of the responder's auxiliary points P_B, Q_B under the isogeny φ_A. The initiator's private key is the pair $\{m_A, n_A\}$ and their public key is the triple $\{E_A, \varphi_A(P_B), \varphi_A(Q_B)\}$.

The responder randomly chooses a pair of integers $m_B, n_B \in \mathbb{Z}/l_B^{e_B}\mathbb{Z}$ and forms the point $R_B = m_B P_B + n_B Q_B \in E\left[l_B^{e_B}\right]$. The responder then computes the isogenous curve E_B corresponding to the isogeny $\varphi_B: E \longrightarrow E_B = E/\langle R_B \rangle$ given by the kernel $\langle R_B \rangle$. Finally, the initiator computes the images $\varphi_B(P_A)$ and $\varphi_B(Q_A)$ of the initiator's auxiliary points P_A, Q_A under the isogeny φ_B. The responder's private key is the pair $\{m_B, n_B\}$ and their public key is the triple $\{E_B, \varphi_B(P_A), \varphi_B(Q_A)\}$.

NOTE: Various compression techniques have been proposed which reduce the size of the SIDH public keys, but in some cases increase the computational cost for both parties. These are briefly described in clause B.1.3.

7.2.4 Key exchange

The initiator and responder exchange public keys and use them, together with their private keys, to derive a shared secret key. Again, the key derivation is slightly different for the initiator and responder.

The initiator uses their private values m_A, n_A and the responder's public points $\varphi_B(P_A), \varphi_B(Q_A)$ to compute the point:

$$m_A\varphi_B(P_A) + n_A\varphi_B(Q_A) = \varphi_B(m_A P_A) + \varphi_B(n_A Q_A) = \varphi_B(m_A P_A + n_A Q_A) = \varphi_B(R_A).$$

The initiator then computes the isogenous curve E_{AB} corresponding to the isogeny $\varphi_{AB}: E_B \longrightarrow E_{AB} = E_B/\langle \varphi_B(R_A) \rangle$ given by the kernel $\langle \varphi_B(R_A) \rangle \subset E_B$. Finally, the initiator derives the shared secret key K from $j(E_{AB})$, the j-invariant of the curve E_{AB}.

The responder uses their private values m_B, n_B and the initiator's public points $\varphi_A(P_B), \varphi_A(Q_B)$ to compute the point:

$$m_B\varphi_A(P_B) + n_B\varphi_A(Q_B) = \varphi_A(m_B P_B) + \varphi_A(n_B Q_B) = \varphi_A(m_B P_B + n_B Q_B) = \varphi_A(R_B).$$

The initiator then computes the isogenous curve E_{BA} corresponding to the isogeny $\varphi_{BA}: E_A \longrightarrow E_{BA} = E_A/\langle \varphi_A(R_B) \rangle$ given by the kernel $\langle \varphi_A(R_B) \rangle \subset E_A$. Finally, the responder derives the shared secret key K from $j(E_{BA})$, the j-invariant of the curve E_{BA}.

Note that $\varphi_B(R_A)$ is the image of the initiator's kernel generator R_A in the responder's curve E_B and $\varphi_A(R_B)$ is the image of the responder's kernel generator R_B in the initiator's curve E_A. This means that the curve E_{AB} computed by the initiator and the curve E_{BA} computed by the responder are isomorphic and so they will have the same j-invariant.

7.3 Implementation considerations

7.3.1 Static key exchanges

It is well known that static ECDH key exchanges can be vulnerable to invalid public key attacks [i.6] and the same is true for SIDH. Galbraith et al [i.48] describe an active attack in which a malicious responder modifies the points in a valid public key and observes whether this causes the key exchange to fail. Each exchange reveals some information about the initiator's private key and the attacker can recover a full static key with at most $\log_2(p)/2$ exchange attempts.

The proposal by Costello et al. [i.47] claims that public key validation can be performed directly by checking that the curve is isogenous to the base curve E and that the points are independent and have the correct order. However, this form of validation does not prevent the attack in [i.48] as the points are not necessarily the correct images under the isogeny. Indeed, [i.49] shows that the ability to distinguishing valid SIDH public keys from invalid ones is enough to recover the private key. For active security, it is necessary to validate the public keys indirectly and [i.48] suggests using the variant of the Fujisaki-Okamoto transform described in [i.50]. See clause B.1.3 for more details.

7.3.2 Side-channel protection

There has been little research into side-channel vulnerabilities in implementations of the SIDH key exchange. Galbraith et al [i.48] suggest a potential attack against SIDH that recovers the shared secret key from a previous exchange, but this assumes a hypothetical side-channel to leak partial information about the j-invariants. There are also fault injection attacks [i.51] and [i.52] that can recover the initiator's static private key by inducing a fault in either the input to an isogeny computation or the computation itself. These have been adapted from techniques used successfully against other elliptic curve cryptosystems and apply even when the indirect key validation from [i.50] is used. It is possible that other elliptic curve side-channel approaches could similarly be adapted to work against SIDH key exchanges.

7.4 Parameter selection

7.4.1 Proposed parameters

Table 6 lists parameters that have been proposed for the SIDH key exchange together with the corresponding public key sizes and claimed security levels. The compressed public keys use the techniques from [i.53].

Table 6: Proposed parameter sets for the SIDH key exchange

Scheme	p	Public key length:		Security level:		Notes
		Uncompressed	Compressed	Classical	Quantum	
Jao-De Feo	$2^{258}\,3^{161}\,186 - 1$	523 bytes	229 bytes	130 bits	86 bits	[i.45]
Jao-De Feo	$2^{341}\,3^{218}\,3 - 1$	670 bytes	302 bytes	172 bits	114 bits	[i.45]
Jao-De Feo	$2^{386}\,3^{242}\,2 - 1$	772 bytes	338 bytes	192 bits	128 bits	[i.45]
Jao-De Feo	$2^{514}\,3^{323}\,353 - 1$	1 036 bytes	453 bytes	258 bits	172 bits	[i.45]
SIDH	$2^{372}\,3^{239} - 1$	752 bytes	329 bytes	187 bits	125 bits	[i.47] and note
NOTE: The SIDH key exchange described in [i.47] uses a form of compression which reduces the size of the public key to 564 bytes without affecting the performance.						

7.4.2 Security estimates

The claimed security levels given in table 6 are based on the estimate that the cost of recovering the private isogeny for an n-bit prime p is approximately $n/4$ bits of classical work and $n/6$ bits of quantum work. Clause B.2 contains a more detailed discussion of the security of the SIDH key exchange.

7.5 Performance

7.5.1 Performance on a 64-bit desktop processor

Table 7 gives performance estimates (in millions of clock cycles) for an implementation of the SIDH key exchange on a 64-bit Intel® Core™ i7 (Haswell) desktop processor. Figures are provided for the key exchange using the partial public key compression from [i47] and the full public key compression from [i.53]. The first performance column refers to the initiator's key generation (step 1 in clause 7.2.1); the second column refers to the responder's key generation and exchange computation (steps 2 and 3); and the third column refers to the initiator's key exchange computation (step 4).

Table 7: Performance of the SIDH key exchange on a 64-bit desktop processor

Scheme	p	Performance:			Public key	Notes
		Initiator (1)	Responder (2-3)	Initiator (4)		
SIDH	$2^{372}\,3^{239} - 1$	90 M cycles	204 M cycles	90 M cycles	564 bytes	[i.53] and note 1
SIDH	$2^{372}\,3^{239} - 1$	205 M cycles	375 M cycles	122 M cycles	329 bytes	[i.53] and note 2
NOTE 1: The quoted performance estimates are for SIDH with partially compressed public keys. NOTE 2: The quoted performance estimates are for SIDH with fully compressed public keys.						

7.5.2 Performance on a 64-bit embedded processor

Table 8 gives performance estimates (in millions of clock cycles) for an implementation of the SIDH key exchange on a single board computer containing a 64-bit ARM® Cortex®-A57 core that has been optimized to use the NEON instruction set.

Table 8: Performance of the SIDH key exchange on a 64-bit embedded processor

Scheme	p	Performance:			Public key	Notes
		Initiator (1)	Responder (2-3)	Initiator (4)		
SIDH	$2^{372}3^{239} - 1$	103 M cycles	231 M cycles	97 M cycles	564 bytes	[i.54]
SIDH	$2^{486}3^{301} - 1$	201 M cycles	459 M cycles	188 M cycles	726 bytes	[i.54]

7.5.3 Performance on a 32-bit embedded processor

Table 9 gives performance estimates (in millions of clock cycles) for an implementation of the SIDH key exchange on a single board computer containing a 32-bit ARM® Cortex®-A15 core that has been optimized to use the NEON instruction set.

Table 9: Performance of the SIDH key exchange on a 32-bit embedded processor

Scheme	p	Performance:			Public key	Notes
		Initiator (1)	Responder (2-3)	Initiator (4)		
SIDH	$2^{250}3^{159} - 1$	83 M cycles	155 M cycles	66 M cycles	377 bytes	[i.55]
SIDH	$2^{372}3^{239} - 1$	437 M cycles	849 M cycles	346 M cycles	564 bytes	[i.55]
SIDH	$2^{501}3^{316}41 - 1$	603 M cycles	1 141 M cycles	516 M cycles	756 bytes	[i.55]

7.6 Summary

Isogeny-based cryptography is still relatively new and the key exchange is essentially the first concrete proposal for an isogeny-based primitive.

The SIDH key exchange has small public keys, particularly when additional compression techniques are used, but its performance is not as good as other quantum-safe key exchanges. It could be a suitable drop-in replacement for DH or ECDH key exchanges in applications where performance is less important than key sizes or where highly-optimized implementations are possible. If it is used to replace static DH or ECDH key exchanges then it can be vulnerable to attacks that exploit invalid public keys unless appropriate countermeasures are taken.

The general problem of recovering an unknown isogeny between two supersingular curves has been relatively well studied, but there has been little analysis of the SIDH key exchange itself or of side-channel vulnerabilities in its implementations. Consequently, more research is needed before a consensus can be established around the security of the SIDH key exchange.

8 Key exchanges from key transport mechanisms

8.1 General construction

In a key transport mechanism, the secret key is generated by the responder and encrypted under the initiator's public key. However, a shared secret key produced by a key exchange is derived from information contributed by both the initiator and responder so that it cannot be completely controlled by either one of them.

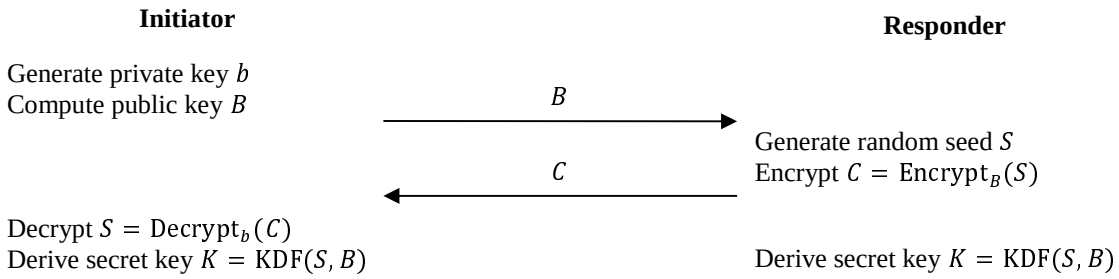

Figure 4: Key exchange from a key transport mechanism

Although not widely used, there are various methods for converting a key transport mechanism into a key exchange. One option, described below and illustrated in figure 4, is to use the public key as one of the inputs to the final Key Derivation Function (KDF):

1) The initiator generates a private key b and computes the corresponding public key B. The initiator sends the public key B to the responder.

2) The responder generates a random seed S and encrypts it using the initiator's public key B to obtain the ciphertext $C = \mathrm{Encrypt}_B(S)$. The responder sends the ciphertext C to the initiator.

3) The responder derives the shared secret key $K = \mathrm{KDF}(S, B)$ from the random seed S and the initiator's public key B.

4) The initiator uses their private key b to decrypt the seed $S = \mathrm{Decrypt}_b(C)$ and derives the shared secret key $K = \mathrm{KDF}(S, B)$ from the decrypted seed S and the public key B.

If the key derivation function is secure then the responder will have no more control over the final shared secret key than in a DH-style key exchange.

Clause 8.2 describes a key exchange based on Niederreiter key transport and clause 8.3 describes a key exchange based on NTRUEncrypt key transport. More examples of quantum-safe key transport mechanisms can be found in [i.2].

8.2 Niederreiter

8.2.1 Introduction

The Niederreiter cryptosystem [i.56] is a code-based key transport mechanism based on McEliece encryption [i.57], one of the first public-key cryptographic algorithms. Its security relies on the difficulty of the decoding problem in a random code and the indistinguishability of the code used in the scheme from a random code. McEliece's proposal to use binary Goppa codes gives very large public keys, but most of the attempts to reduce key sizes by introducing different code families have been broken over time. The original proposal has withstood analysis for almost 40 years and has become a trusted candidate for post-quantum cryptography.

Clause 8.2.2 gives a description of the Niederreiter key exchange, clause 8.2.3 discusses some implementation considerations, clause 8.2.4 lists proposed parameters, and clause 8.2.5 gives performance estimates. Performance comparisons with other key exchanges can be found in annex C.

8.2.2 Niederreiter key exchange

8.2.2.1 Overview

The key exchange is obtained from the Niederreiter key transport using the conversion described in clause 8.1. The initiator's public key is a parity check matrix for a binary Goppa code that has been scrambled to hide the structure in the code. The private key consists of the information used to construct the Goppa code and the scrambling matrix. The responder selects a random error vector and uses the public parity matrix to compute the corresponding syndrome. The initiator knows the scrambling of the public key and the structure of the code and can therefore decode the syndrome to recover the error vector. The shared secret key is derived from the error vector and public parity check matrix.

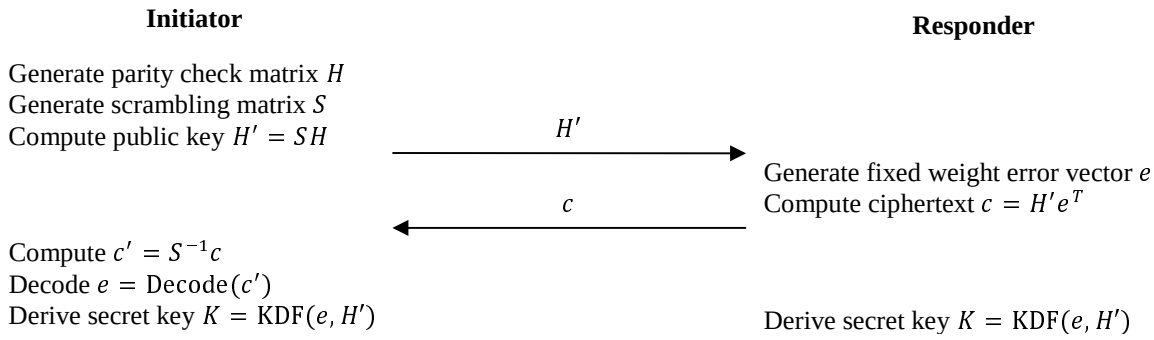

Figure 5: Niederreiter key exchange

More specifically, the Niederreiter key exchange consists of the following steps, as illustrated in figure 5:

1) The initiator generates a random binary Goppa code with private parity check matrix H and computes the public key $H' = SH$ where S is a random invertible binary matrix. The initiator sends the public key H' to the responder.

2) The responder generates a random binary error vector e of fixed length and Hamming weight, and computes the ciphertext $c = H'e^T$. The responder sends the ciphertext c to the initiator.

3) The responder derives the shared secret key $K = \mathrm{KDF}(e, H')$ from the error vector e and the public key H'.

4) The initiator descrambles the ciphertext by multiplying it by the inverse of S to obtain $S^{-1}c = He^T$ and uses the private Goppa code structure to decode the error vector e. The initiator then derives the shared secret key $K = \mathrm{KDF}(e, H')$ from the decoded error vector e and the public parity check matrix H'.

8.2.2.2 Public parameters

The public parameters for the Niederreiter key exchange are:

- The length n of the code;
- The rank k of the code; and
- The Hamming weight t of the error vector.

The binary Goppa code is defined by a square-free polynomial of degree t over a finite field $\mathrm{GF}(2^m)$ and a vector of n distinct elements from $\mathrm{GF}(2^m)$ that are not roots of the polynomial. For the security of the scheme, it is important that both the Goppa polynomial and the support vector are kept private.

The Goppa code can correct t errors and is usually chosen so that it has full rank $k = n - mt$ where $m = \lceil \log_2(n) \rceil$. The number of correctable errors can be increased by using list decoding techniques.

The error vector e is a random binary vector of length n and Hamming weight t and the ciphertext $c = H'e^T$ is a binary vector of length $n - k$.

8.2.2.3 Key generation

In practice, the Goppa polynomial is usually assumed to be irreducible since Shoup's algorithm [i.58] can be used to efficiently generate a uniformly random irreducible polynomial of degree t.

The scrambling matrix S can often be chosen so that the $(n - k) \times n$ public parity check matrix H' is in standard form; that is, $H' = [I_{n-k} \quad H'']$ where I_{n-k} is the $(n - k) \times (n - k)$ identity matrix. This can always be done after swapping some of the columns of H, together with corresponding entries of the support vector. The effective size of the public key can therefore be reduced by insisting that H' is in standard form and only sending the $(n - k) \times k$ binary matrix H''.

8.2.2.4 Decryption

Binary Goppa codes can be efficiently decoded by Patterson's algorithm [i.59] or Berlekamp's algorithm [i.60]. Both algorithms will decode *t* errors in a binary Goppa code. Patterson's algorithm seems to be somewhat faster when not decoding more than *t* errors, while Berlekamp's algorithm can be implemented more compactly.

8.2.3 Implementation considerations

8.2.3.1 Active attacks

In general, the McEliece and Niederreiter encryption schemes are vulnerable to a range of active attacks so need to be used with an appropriate conversion, such as [i.61], to provide semantic security. This is not necessary for the Niederreiter key exchange as the same attacks do not apply in this case. Persichetti [i.62] describes an actively secure Key Encapsulation Mechanism (KEM) that is very similar to the key exchange in clause 8.2.2. The main difference between the two schemes is that Persichetti's KEM returns a pseudo-randomly chosen secret key when there is a decoding failure.

NOTE: Although the modified KEM in [i.62] has a security proof in the random oracle model, it is not clear that it is secure against attacks that exploit decoding failures [i.63] since it would still be possible for a malicious responder to detect a failure. This type of attack is not a concern when using binary Goppa codes.

8.2.3.2 Side-channel attacks

The side-channel security of code-based cryptography is an active area of research. There are a number of different side-channel attacks on McEliece and Niederreiter decryption [i.64] and [i.65] that exploit timing or power information leaked by the decoding algorithm to recover the private key. Berlekamp's algorithm is easier to protect against side-channel attacks than Patterson's algorithm, but it can still be difficult to eliminate all the side-channel information. For example, although McBits [i.66] claims to implement a constant-time decoding algorithm, [i.67] suggests that it could still be vulnerable to the timing attack described in [i.68].

8.2.4 Parameter selection

8.2.4.1 Proposed parameters

Table 10 lists parameters that have been proposed for the Niederreiter key transport mechanism together with their classical and quantum security estimates.

Table 10: Proposed parameter sets for the Niederreiter key exchange

Scheme	n	k	t	Public key	Ciphertext	Security estimates:		Notes
						Classical	Quantum	
PQCrypto	6 960	5 413	119	1 046 739 bytes	194 bytes	263	≥ 128	[i.69] and note 1
Bernstein et al	2 960	2 288	57	192 192 bytes	84 bytes	128	≥ 64	[i.70] and note 2
Bernstein et al	6 624	5 129	117	958 482 bytes	187 bytes	256	≥ 128	[i.70] and note 3
de Vries	5 542	4 242	100	689 325 bytes	163 bytes	199	128	[i.71]

NOTE 1: These are the initial recommendations from the PQCrypto project [i.69]. The analysis from [i.71] suggests that the parameters have 249 bits of classical security and 153 bits of quantum security.
NOTE 2: These parameters use list decoding [i.72] to increase the number of errors that can be decoded from 56 to 57. The analysis from [i.71] suggests that the parameters have 82 bits of quantum security.
NOTE 3: These parameters use list decoding [i.72] to increase the number of errors than can be decoded from 115 to 117. The analysis from [i.71] suggests that the parameters have 150 bits of quantum security.

8.2.4.2 Security estimates

The Niederreiter key exchange parameters are chosen so that it is infeasible to distinguish the Goppa code from a random binary code and to decode a random code with the same parameters. In practice, the most efficient attacks against Niederreiter use information-set decoding techniques. The estimated cost of classical information-set decoding is well-understood, but quantum approaches have been studied less. It is known that Grover's algorithm can speed up existing decoding attacks [i.71] and [i.73] such that the time-complexity of quantum-decoding attacks is at least the square root of the complexity of classical decoding attacks. Recently, the impact of Grover's algorithm on decoding binary Goppa codes was studied in more detail [i.71], resulting in smaller parameters.

8.2.5 Performance

8.2.5.1 Performance on a 64-bit server processor

Table 11 gives performance estimates (in millions of clock cycles) for an implementation of a hybrid version of McEliece encryption on a 64-bit Intel® Xeon® E3 (Sandy Bridge) server processor.

Table 11: Performance of McEliece encryption on a 64-bit server processor

Scheme	n	k	t	Performance:			Public key	Notes
				Key generation	Encryption	Decryption		
McEliece	2 048	1 696	32	30,9 M cycles	0,049 M cycles	1,09 M cycles	74 264 bytes	[i.74] and note
NOTE: This is an implementation of the hybrid McEliece scheme from [i.75]. The designers estimate that the proposed parameters provide 88 bits of classical security.								

8.2.5.2 Performance on a 64-bit desktop processor

Table 12 gives performance estimates (in millions of clock cycles) for the McBits implementation of Niederreiter encryption on a 64-bit Intel® Core™ i7 (Ivy Bridge) desktop processor.

Table 12: Performance of Niederreiter encryption on a 64-bit desktop processor

Scheme	n	k	t	Performance:			Public key	Notes
				Key generation	Encryption	Decryption		
McBits	2 690	2 018	56	---	---	0,071 M cycles	169 512 bytes	[i.66]
McBits	6 624	5 129	115	---	---	0,297 M cycles	958 482 bytes	[i.66]
McBits	8 192	6 528	128	1 550 M cycles	0,312 M cycles	0,492 M cycles	1 357 824 bytes	[i.76]

8.2.5.3 Performance on an 8-bit microcontroller

Table 13 gives performance estimates (in millions of clock cycles) for an implementation of Niederreiter encryption on an 8-bit Atmel® AVR® XMEGA® microcontroller.

Table 13: Performance of Niederreiter encryption on an 8-bit microcontroller

Scheme	n	k	t	Performance:			Public key	Notes
				Key generation	Encryption	Decryption		
Niederreiter	2 048	1 751	27	---	0,051 M cycles	5,75 M cycles	65 006 bytes	[i.77] and note
NOTE: The parameters are estimated to provide 80 bits of security. The implementation used 173 Kbytes of internal flash memory including the private and public keys.								

8.2.6 Summary

Niederreiter encryption using binary Goppa codes is a well-known and well-trusted quantum-safe cryptosystem. The encryption and decryption operations are fast even on constrained devices, but the public keys are very large and key generation is relatively slow. Consequently, the Niederreiter key exchange could not be used as a general-purpose drop-in replacement for DH and ECDH key exchanges as most protocols would likely need to be adapted to accommodate the larger keys. However, it might be a replacement for static DH or ECDH in applications where very large public keys can be tolerated and where there is a need for a high level of confidence in the long-term security of the parameters.

8.3 NTRU

8.3.1 Introduction

NTRUEncrypt [i.18] is a lattice-based encryption scheme that was first proposed in 1996. The security of NTRUEncrypt relies on the difficulty of finding an unusually short vector in a well-structured NTRU lattice. There have been several updates to the parameters following improvements to lattice techniques, but the scheme has largely resisted twenty years of analysis.

Clause 8.3.2 gives a description of the NTRU key exchange, clause 8.3.3 discusses some implementation considerations, clause 8.3.4 lists proposed parameters, and clause 8.3.5 gives performance estimates. Performance comparisons with other key exchanges can be found in annex C.

8.3.2 NTRU key exchange

8.3.2.1 Overview

The NTRU key exchange is obtained from the NTRUEncrypt key transport mechanism using the conversion described in clause 8.1. The initiator's public key is an element of a polynomial ring with modulus q that is privately constructed as a quotient of two polynomials with small coefficients. Messages are represented by elements of a polynomial ring with smaller modulus p. The responder encrypts a random message by blinding it with the public key and a small noise polynomial. The initiator uses the private construction of the public key to remove the noise and recover the message. The shared secret key is derived from the random message and the initiator's public polynomial.

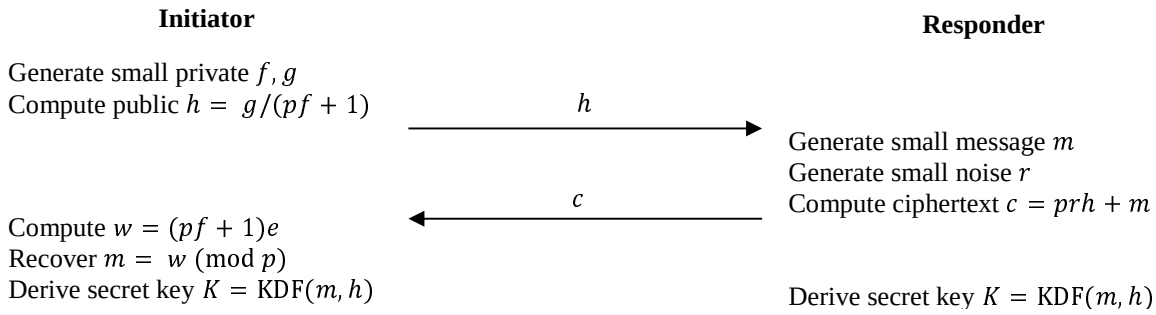

Figure 6: NTRU key exchange

More specifically, the NTRU key exchange consists of the following steps, as illustrated in figure 6:

1) The initiator generates a pair of private polynomials *f* and *g* with small coefficients such that $pf + 1$ is invertible modulo q. The initiator's corresponding public key is the quotient $h = g/(pf + 1)$. The initiator sends the public key *h* to the responder.

2) The responder generates a random message polynomial m and random noise polynomial r with small coefficients and then computes the ciphertext $c = prh + m$. The responder sends the ciphertext *c* to the initiator.

3) The responder derives the shared secret key $K = \mathrm{KDF}(m, h)$ from the random message m and the initiator's public polynomial h.

4) The initiator decrypts the message by computing $w = (pf + 1)c$ and then recovering $m = w \pmod p$. The initiator derives the shared secret key $K = \mathrm{KDF}(m, h)$ from the decrypted message m and the public polynomial h.

8.3.2.2 Public parameters

The public parameters for the NTRU key exchange are:

- A prime n which defines the ring $R = \mathbb{Z}[x]/(x^n - 1)$;
- A large integer q which gives the modulus for the ring $R_q = R/qR$;
- A small integer p which defines the message space; and
- Distributions D_f, D_g, D_r and D_m which are used to sample small polynomials from R.

The large modulus q is typically chosen to be a power of two for efficient arithmetic. The small modulus p is typically chosen to be 3.

The distributions D_f, D_g, D_r and D_m can be the same, but do not need to be. One option is to fix a value d close to $n/3$ and use the uniform distribution over all polynomials in R with d coefficients equal to 1, d coefficients equal to -1, and the remaining coefficients equal to 0. The parameter sets in [i.78] and [i.79] use this type of distribution for D_g, but a more complicated product-form distribution for D_f.

8.3.2.3 Decryption

Given the ciphertext $c = prh + m$, the initiator attempts to decrypt the message by computing the value

$$w = (pf + 1)c = p(rg + fm) + m \pmod q,$$

lifting it to R and reducing modulo p. This can fail if any of the coefficients of error term $p(rg + fm)$ are large enough to wrap-around modulo q.

8.3.3 Implementation considerations

8.3.3.1 Static key exchange

There are various active attacks against NTRUEncrypt [i.80], [i.81] and [i.82] in which a malicious responder uses decryption failures to recover the initiator's full static private key. In [i.80] and [i.82] the attacker needs to be able to see the incorrectly decrypted message so these attacks do not apply to the NTRU key exchange. However, in [i.81] the attacker only needs to know that decryption has failed which means that this attack could be used against the key exchange.

The NTRUEncrypt parameters from [i.79] are chosen so that the probability of a decryption failure with a message chosen from the correct distribution D_m is at most $2^{-\lambda}$, where λ is the security parameter. NTRU-KEM [i.83] is more conservative and chooses the parameters so that decryption never fails when the messages are chosen from D_m.

Different techniques have been suggested to prevent a malicious responder from using malformed ciphertexts to force a decryption failure. The actively secure NTRUEncrypt described in [i.84] includes a randomized padding scheme which reduces the responder's control over the construction of the ciphertext. The actively secure NTRU-based KEMs in [i.83] and [i.85] use variants of the Fujisaki-Okamoto transform which allow the initiator to verify that the responder constructed the ciphertext honestly.

8.3.3.2 Side channel attacks

An introduction to side-channel protection for lattice-based cryptography is provided in [i.32]. NTRUEncrypt is vulnerable to both timing [i.86] and power analysis [i.87] attacks that exploit information leaked during the decryption process. Reference [i.83] introduces an efficient constant-time implementation of NTRU-KEM which should give protection against timing attacks. However, it has been harder to produce countermeasures against power analysis techniques [i.88] and [i.89].

8.3.4 Parameter selection

8.3.4.1 Proposed parameters

Table 14 lists parameters that have been proposed for NTRU-based key transport mechanisms with their classical and quantum security estimates. A process for generating additional parameter sets is described in [i.79].

Table 14: Proposed parameter sets for the NTRU key exchange

Scheme	n	k	p	Public key length	Security estimates:		Decryption failure	Notes
					Classical	Quantum		
NTRUEncrypt	443	2 048	3	610 bytes	128	128	2^{-128}	[i.78] and note 1
NTRUEncrypt	587	2 048	3	808 bytes	192	128	2^{-192}	[i.78] and note 1
NTRUEncrypt	743	2 048	3	1 022 bytes	256	128	2^{-256}	[i.78] and note 1
NTRU-KEM	701	8 192	3	1 140 bytes	217	136	0	[i.83] and note 2
NOTE 1: The quantum security estimates given in [i.78] are based on the hash functions used in key generation and message encoding rather than the cost of the quantum hybrid attack. NOTE 2: NTRU-KEM decryption can never fail for validly chosen messages.								

8.3.4.2 Security estimates

In practice, the most efficient attack against NTRUEncrypt is the hybrid attack from [i.90] which combines lattice reduction and a meet-in-the-middle search. The classical security analysis in [i.79] and [i.83 assumes that the lattice reduction is performed using the Block Korkine-Zolotarev (BKZ) algorithm with pruned enumeration rather than the asymptotically more efficient lattice sieving. This means that the security estimates in table 14 will be less conservative than the corresponding classical estimates for the LWE and Ring-LWE key exchanges given in clause A.2.2. The quantum version of the hybrid attack considered in [i.83] uses quantum enumeration or quantum sieving to improve the efficiency of lattice reduction and replaces the meet-in-the-middle phase with a quantum search.

8.3.5 Performance

8.3.5.1 Performance on a 64-bit desktop processor

Table 15 gives performance estimates (in millions of clock cycles) for implementations of NTRUEncrypt and NTRU-KEM on a 64-bit Intel® Core™ i7 (Haswell) desktop processor.

Table 15: Performance of NTRUEncrypt and NTRU-KEM on a 64-bit desktop processor

Scheme	n	q	Performance:			Public key	Notes
			Key generation	Encryption	Decryption		
NTRUEncrypt	439	2 048	0,452 M cycles	0,048 M cycles	0,050 M cycles	604 bytes	[i.74]
NTRUEncrypt	593	2 048	0,697 M cycles	0,063 M cycles	0,065 M cycles	816 bytes	[i.74]
NTRUEncrypt	743	2 048	0,974 M cycles	0,083 M cycles	0,090 M cycles	1 022 bytes	[i.74]
NTRU-KEM	701	8 192	0,308 M cycles	0,049 M cycles	0,067 M cycles	1 140 bytes	[i.83]

8.3.5.2 Performance on a 32-bit embedded processor

Table 16 gives performance estimates (in millions of clock cycles) for an implementation of NTRUEncrypt on a single board computer containing a 32-bit ARM® Cortex®-A8 core.

Table 16: Performance of NTRUEncrypt on a 32-bit embedded processor

Scheme	n	q	Performance:			Public key	Notes
			Key generation	Encryption	Decryption		
NTRUEncrypt	439	2 048	10,8 M cycles	0,294 M cycles	0,528 M cycles	604 bytes	[i.74]
NTRUEncrypt	593	2 048	19,4 M cycles	0,454 M cycles	0,817 M cycles	816 bytes	[i.74]
NTRUEncrypt	743	2 048	30,4 M cycles	0,685 M cycles	1,29 M cycles	1 022 bytes	[i.74]

8.3.5.3 Performance on a 32-bit microcontroller

Table 17 gives performance estimates (in millions of clock cycles) for an implementation of NTRUEncrypt on a 32-bit ARM® Cortex®-M0 microcontroller.

Table 17: Performance of NTRUEncrypt on a 32-bit microcontroller

Scheme	n	q	Performance:			Public key	Notes
			Key generation	Encryption	Decryption		
NTRUEncrypt	443	2 048	26,1 M cycles	0,588 M cycles	0,950 M cycles	610 bytes	[i.91] and note 1
NTRUEncrypt	587	2 048	45,7 M cycles	1,04 M cycles	1,63 M cycles	808 bytes	[i.91] and note 2
NTRUEncrypt	743	2 048	71,2 M cycles	1,41 M cycles	2,38 M cycles	1 022 bytes	[i.91] and note 3
NOTE 1: The implementation used around 5,6 Kbytes of internal flash memory. NOTE 2: The implementation used around 5,8 Kbytes of internal flash memory. NOTE 3: The implementation used around 5,9 Kbytes of internal flash memory.							

8.3.6 Summary

NTRUEncrypt was one of the first practical lattice-based cryptosystems and it has received considerable analysis. Its public keys and ciphertexts are small and the performance of encryption and decryption is good, but key generation is considerably slower than for the Ring-LWE key exchanges. Consequently, the NTRU key exchange could be used as a drop-in replacement for ephemeral DH and ECDH key exchanges in applications where performance is not critical. If it is used to replace static DH or ECDH key exchanges then it can be vulnerable to attacks that exploit decryption failures unless appropriate countermeasures are taken.

9 Conclusions

A straightforward approach to transitioning from currently deployed public-key cryptography to quantum-safe cryptography is to substitute the existing primitives with like-for-like quantum-safe "drop-in replacements". This assumes that suitable alternatives with similar security levels and efficiency properties are available. For most network security applications, the immediate quantum threat is to the confidentiality rather than the authentication.

ETSI GR QSC 003 [i.5] provides some examples of typical use cases for quantum-safe key exchanges together with an extended discussion of practical implementation issues. These include integration into the protocol stack where the larger public key sizes of most quantum-safe primitives will likely require more attention being paid to packet fragmentation and handshake time-outs.

The present document considers in more detail a selection of proposals for quantum-safe key exchanges taken from the academic literature. The LWE, Ring-LWE and SIDH key exchanges are the closest to being drop-in replacements for DH or ECDH, although their public keys are all still somewhat larger than in existing key exchanges and they need to be protected against active attacks.

The lattice-based key exchanges considered in clause 6 have been extensively studied in the academic literature. They offer a good balance between security and efficiency, particularly when using structured lattices. However, as is highlighted in clause A.1, there are a variety of implementation details that will need care to get right.

The SIDH key exchange considered in clause 7 offers the most compact parameters, but at the expense of additional computation. It is based on advanced mathematics that could be difficult for non-experts to implement. The main drawback is that it is still a relatively new proposal and, as noted in ETSI GR QSC 001 [i.2], more research will be needed for a consensus to be established around its security.

Other non-standard constructions for key exchanges are examined in clause 8. Key exchanges based on RSA are not in common use today, but where such construction are being used a key exchange built from a quantum-safe key transport mechanism such as Niederreiter or NTRU could provide a good quantum-safe replacement. The security and efficiency properties of quantum-safe key transport mechanisms will be examined in greater detail in a separate document.

Finally, there have also been proposals for hybrid key exchanges in which the shared secret key is derived from a combination of the outputs from a classical DH or ECDH key exchange and from a separate, and perhaps novel, quantum-safe key exchange. This might be a viewed as an intermediate step in the transition to using purely quantum-safe cryptography, or as a way of providing extra functionality or security. Hybrid schemes are discussed further in ETSI GR QSC 003 [i.5].

Annex A:
LWE design and security considerations

A.1 LWE and Ring-LWE variants

A.1.1 Rings

Ring-LWE, and Polynomial-LWE more generally, is based on the polynomial ring $R = \mathbb{Z}[x]/(f(x))$ and its quotient $R_q = (\mathbb{Z}/q\mathbb{Z})[x]/(f(x))$ for some choice of irreducible polynomial $f(x) \in \mathbb{Z}[x]$ and prime modulus $q \in \mathbb{Z}$.

Most concrete Ring-LWE proposals, including BCNS [i.28] and New Hope [i.27], choose a cyclotomic polynomial $f(x) = x^{m/2} + 1$ where m is a power of two and a modulus $q \equiv 1 \pmod{m}$. The advantage of this polynomial and modulus is that arithmetic in R_q can be implemented very efficiently using a special version of the Number-Theoretic Transform (NTT). However, using power-of-two cyclotomic polynomials severely limits the choice of parameters so it can lead to larger public key sizes than are necessary to achieve a desired level of security.

Cyclotomic polynomials $f(x) = x^{m-1} + x^{m-2} + \cdots + x + 1$ where m is prime and $q \equiv 1 \pmod{m}$ offer a greater range of parameter choices and so potentially smaller public keys, but only allow a less efficient version of the NTT for the arithmetic. Comparing the results in [i.39] and [i.40] suggests that key exchanges using prime cyclotomic polynomials are about 50 % slower than those using power-of-two cyclotomic polynomials, but have public keys that are 10 to 20 % smaller for an equivalent security level.

There have been suggestions that the subfield structure of cyclotomic rings could be used to improve key recovery attacks against NTRUEncrypt-style schemes [i.92] and [i.93]. NTRU Prime [i.94] is a proposal which uses a polynomial of the form $f(x) = x^m - x - 1$ where m is a prime and choosing q so that R_q is a prime field. The NTT cannot be used with such rings so NTRU Prime relies on a combination of Karatsuba and Toom-Cook for the polynomial arithmetic. The NTRU Prime designers claim that this can be made nearly as efficient as arithmetic using the power-of-two NTT, but it is not possible to compare NTRU Prime and New Hope directly as [i.94] does not give performance estimates for the full cryptographic operations. The attacks in [i.92] and [i.93] have been superseded by a geometric attack on NTRU [i.95] that does not need subfields and so applies to NTRU Prime.

A.1.2 Distributions

A.1.2.1 Discrete Gaussians

The original definition of the LWE problem [i.19] assumed that the entries of the private key were sampled uniformly from $\mathbb{Z}/q\mathbb{Z}$ and that the error terms were sampled from a one-dimensional Gaussian distribution. The description of the key exchange in clause 6.2 uses normal-form LWE [i.96] which allows the private keys to be sampled from the error distribution while retaining the theoretical security guarantees.

In Ring-LWE schemes the coefficients of the private keys and noise terms are properly sampled from a spherical discrete Gaussian [i.21]. For power-of-two cyclotomic rings, such as in New Hope [i.27], this is equivalent to sampling each coefficient independently from a one-dimensional discrete Gaussian. For prime cyclotomic rings, such as in Singh-Chopra [i.40], sampling the coefficients independently distorts the distribution so that it is no longer spherical. There are approaches that can be taken to compensate for the distortion in this case [i.97]. However, in more general rings some care needs to be taken to avoid choosing a weak distribution for that ring [i.98].

A.1.2.2 Approximate Gaussians

Sampling accurately from discrete Gaussians can be expensive, but for the LWE and Ring-LWE based key exchanges it is possible to use approximate distributions without weakening the practical security of the scheme.

- Frodo [i.25] improves the efficiency of key generation by sampling from a simpler distribution on a bounded interval which is still relatively close to a discrete Gaussian.

- The performance estimates in [i.39] suggest that key generation using bounded uniform distributions can be 5-6 times faster than using a discrete Gaussian.
- Key generation in New Hope [i.27] using a centred binomial distribution is almost 10 times faster than key generation in BCNS [i.28] using a discrete Gaussian, although this could partially be due to improvements in the polynomial arithmetic.

A.1.2.3 Small distributions

Some LWE and Ring-LWE schemes sample private key and error terms from small distributions such as the uniform binary distribution over $\{0,1\}$, the uniform ternary distribution over $\{-1,0,1\}$, or the ternary distribution with a fixed number of nonzero terms. These distributions lead to efficient arithmetic, lower memory requirements and improved failure rates. Consequently, small distributions have been included in schemes proposed for LWE encryption [i.99], LWE key exchange [i.26], Ring-LWE identity-based encryption [i.100] and Ring-LWE fully homomorphic encryption [i.101].

Micciancio and Peikert [i.102] show that, under some assumptions on the number of samples, small distribution LWE retains a worst-case hardness guarantee. However, the practical security is affected by attacks such as [i.103] and [i.104] which exploit the small distribution. Consequently, these attacks need to be taken into consideration when choosing parameters for a LWE or Ring-LWE scheme that uses small distributions.

A.1.2.4 Learning with Rounding

Learning with Rounding (LWR) [i.105] is an alternative method of introducing error terms in LWE by rounding to a modulus p that is smaller than the LWE modulus q. The resulting error terms are deterministic, rounding reduces the size of the public keys that need to be transmitted, and some of the arithmetic can be performed with respect to the smaller modulus. It has been shown [i.105] that, under some assumptions on the number of samples, LWR is at least as hard as LWE. Lizard [i.106] is a public-key encryption scheme based on both the LWE and LWR problems that uses small distributions. The spKEX key exchange [i.26] relies on LWR.

A.1.3 Varying A

The general description of the Ring-LWE key exchange in [i.24], and its instantiation in [i.28], uses a fixed public parameter A. The designers of Frodo [i.25] and New Hope [i.27] raise two related security concerns about using this approach:

- Fixing the parameter A means that every key exchange will involve the same lattice. Performing a single large computation on this lattice could then allow any private key to be recovered with a small amount of secondary work.
- If the fixed choice of A corresponds to a weak instance of the lattice problem then the security of all users of the system will be affected.

As a consequence, Frodo and New Hope both treat A as part of the initiator's public key and generate a fresh value for each key exchange. This ensures that every exchange involves a different lattice and the large lattice computation will need to be repeated for each key recovery attack.

To reduce bandwidth, Frodo and New Hope pseudo-randomly generate A from a small seed so that only the seed needs to be included in the initiator's public key. However, there are disadvantages to doing this:

- Pseudo-random generation incurs a performance overhead for both the initiator and generator. The performance estimates for Frodo given in [i.25] suggest that expanding the public matrix A from a seed accounts for almost half of the time taken to generate a new key pair. It is important to note that these figures depend on hardware support for AES. When hardware support is not available, pseudo-random generation will be significantly more expensive.
- Pseudo-randomly generating A from a small seed rather than sampling it uniformly from R_q invalidates the security proof for the key exchange from [i.24]. A modified proof incorporating the pseudo-random generation is described in [i.25], but this no longer gives a tight security bound and the authors indicate that it might not hold against quantum adversaries.

In some situations, it might be necessary to find a compromise between generating a fresh A for each key exchange and using a fixed parameter A in all exchanges. The designers of Frodo [i.25] suggest that A could be cached and reused for a limited time, but this would only reduce the computational burden for the initiator and not the responder. Another option used in [i.26] is to have a master public parameter A generated by a trusted party which is then modified by the initiator in an easily computable way to derive a new public parameter A' for each exchange. This ensures that there is sufficient randomness in the choice of A' while minimizing the computational requirement for both the initiator and the responder.

A.1.4 Reconciliation mechanisms

The reconciliation mechanism for the LWE key exchange described in clause 6.2 returns the correct shared secret key when the entries of the error term $V_r - V_i = S_r \cdot E_i - E_r \cdot S_i + E'_r$ are less than $q/2^{B+2}$ in absolute value. Similarly, the Ring-LWE reconciliation mechanism in clause 6.3 returns the correct shared secret key when the coefficients of the error term $V_r - V_i = s_r \cdot e_i - s_i \cdot e_r + e'_r$ are less than $q/8$ in absolute value. Increasing the modulus q or decreasing the standard deviation σ improves the probability that the key exchange will succeed, but reduces the security of the corresponding lattice problem. Parameter selection therefore involves finding an appropriate balance between the security and correctness of the scheme.

New Hope [i.27] is able to significantly improve this balance by using a more sophisticated reconciliation mechanism adapted from ideas given in [i.107]. Each bit of the shared secret key is extracted from four consecutive coefficients of the intermediate value using Voronoi cell decoding in a four-dimensional lattice. Although more complicated to implement and analyse, this mechanism returns the correct shared secret key when the sums of the absolute values of the four consecutive error coefficients are at most $3q/4$. This allows the designers of New Hope to reduce the size of the modulus, and so increase the security of the scheme, while still having an acceptable failure probability.

An approach to improving the Frodo reconciliation mechanism is proposed in [i.108]. It increases the number of check bits that are sent for each entry of the intermediate matrix V_r so that more key bits can be extracted without significantly affecting the failure probability. This means that smaller dimensional matrices can be used to establish a shared secret key of the same length leading to reductions in the bandwidth requirements of up to 20 %. This method is used in [i.26].

A.1.5 Key transport

The first proposals for public-key algorithms based on LWE and Ring-LWE focused on encryption schemes rather than key exchange. It is straightforward to modify the key exchanges in clauses 6.2 and 6.3 to give the corresponding key transport mechanisms. The present clause describes this for Ring-LWE and the changes for the LWE scheme are similar.

Key generation for the Ring-LWE key transport is identical to the Ring-LWE key exchange.

1) The initiator generates a private key s_i and private noise term e_i, and computes the corresponding public key $B_i = s_i \cdot A + e_i$. The initiator sends their public key B_i to the responder.

2) The responder generates a private key s_r and private noise term e_r, and computes the corresponding public key $B_r = s_r \cdot A + e_r$.

The responder forms the same intermediate value $s_r \cdot B_i + e'_r$ as before, but this time it is used to blind the secret key K_r. The secret key is viewed as a polynomial with coefficients in $\{0,1\}$.

3) The responder generates a second noise term e'_r and computes the value $C = s_r \cdot B_i + e'_r + \mathrm{Floor}(q/2) \cdot K_r$. The responder sends their public key B_r and the blinded secret key C to the initiator.

The initiator can then use their intermediate value $s_i \cdot B_r$ to unblind the secret key.

4) The responder forms the value $W = C - s_i \cdot B_r$ and recovers the secret key as $K_i = \mathrm{Round}(2W/q) \bmod 2$. The secret keys K_i and K_r will be the same with reasonable probability.

NOTE: The bandwidth requirements can be reduced by only sending the top few bits of each coefficient of C. This increases the size of the error in the final decryption step so the parameters will need to be adjusted to maintain an acceptable decryption failure rate.

A.2 Security considerations

A.2.1 Provable security

Let $s \in (\mathbb{Z}/q\mathbb{Z})^n$ be a fixed vector and χ be a discrete Gaussian distribution over $\mathbb{Z}$. The search LWE problem is to find s given a sequence of LWE samples of the form (a_i, b_i), where the a_i are chosen uniformly at random from $(\mathbb{Z}/q\mathbb{Z})^n$ and $b_i = \langle a_i, s\rangle + e_i \pmod q$ for some e_i chosen from χ. The decisional version is to distinguish a sequence of LWE samples from a sequence of pairs (a_i, b_i) where the b_i are chosen uniformly at random from $\mathbb{Z}/q\mathbb{Z}$. An oracle for the decisional problem can be used to solve the search problem in a classical algorithm with a polynomial number of steps.

Regev [i.19] showed that there is a quantum reduction to the decisional LWE problem, with appropriate parameters, from a decisional approximate short vector problem on arbitrary n-dimensional lattices where the approximation factor depends on the ratio between the modulus q and the standard deviation of χ. Brakerski et al [i.109] have subsequently produced a classical reduction from an approximate decisional short vector problem on arbitrary $\sqrt{n}$-dimensional lattices.

The search and decisional Ring-LWE problems are analogous. Lyubashevsky, Peikert and Regev [i.21] have shown that there is a quantum reduction from the decisional Ring-LWE problem, with appropriate parameters, to a computational approximate short vector problem on arbitrary ideal lattices in R. No classical reduction is known in this case.

Although the reductions provide some level of confidence in the security of the LWE and Ring-LWE key exchanges, [i.110] notes that the lack of tightness in the results means that they are not useful when choosing practical parameters. Instead, parameters for the key exchanges are typically chosen based on estimates of the difficulty of the LWE and Ring-LWE problems themselves.

Reference [i.24] contains a more detailed discussion of the LWE and Ring-LWE problems.

A.2.2 Passive security

The passive security of the LWE and Ring-LWE key exchanges is largely determined by the difficulty of solving certain close or short vector problems. There have been a series of attacks [i.111], [i.112], [i.113] and [i.114] against schemes over polynomial rings that were specifically chosen to be weak, but these are not relevant to Ring-LWE when used with cyclotomic rings and correctly chosen distributions [i.98].

The two main lattice attacks [i.115] are:

- Decoding attack: Recover the private key by finding a vector in a lattice of the form:

$$L = \{(x, y) \in \mathbb{Z}^m \times \mathbb{Z}^n \mid x \equiv A \cdot y \pmod q\}$$

  that is sufficiently close to a target vector corresponding to the public key. This can be rephrased as a unique short vector problem by embedding L into a higher-dimensional lattice that includes the target vector.

- Distinguishing attack: Distinguish the public key from random by finding vectors in the scaled dual lattice:

$$L' = \{(x, y) \in \mathbb{Z}^m \times \mathbb{Z}^n \mid x \cdot A \equiv y \pmod q\}$$

  that are sufficiently short.

For New Hope [i.27] and Frodo [i.25] the authors determine the block size needed to produce vectors of the required length using BKZ and then estimate the cost of the attack as the asymptotic cost of lattice sieving or quantum lattice sieving on a block of that size. This type of analysis is simple to perform and gives a conservative lower bound on the security of the parameters, but could lead to unnecessarily large parameter sizes.

The concrete security analysis in [i.41] is more complex as it attempts to find the best attack from a wider range of approaches including small secret variations and non-lattice attacks such as Blum-Kalai-Wasserman [i.116]. Further, [i.41] also tries to give a more realistic assessment of the overall cost of the attack using state-of-the-art lattice algorithms. This analysis is likely to produce tighter security estimates, but will be affected more by advances in lattice techniques.

Table A.1 compares the security estimates produced by the two approaches for the key exchange parameters listed in clause 6.5. The concrete security estimates were produced by the estimator (version f59326c) from [i.41] using the same classical and quantum lattice sieving cost models as in [i.27].

Table A.1: Security comparisons for the LWE and Ring-LWE proposed parameters

Scheme	n	q	σ	Concrete estimate:		Conservative estimate:		Failure probability
				Classical	Quantum	Classical	Quantum	
Frodo	752	2^{15}	1,32	170	156	144	130	2^{-39}
spKEX	738	2^{14}	2,309	---	---	141	128	2^{-42}
BCNS	1 024	$2^{32} - 1$	3,192	98	90	86	78	$2^{-131072}$
Singh	1 024	40 961	3,192	294	266	248	225	2^{-91}
New Hope	1 024	12 289	2,828	337	306	281	255	2^{-60}
Singh-Chopra	540	41 117	3,192	138	128	114	103	2^{-87}

A.2.3 Active security

Peikert [i.24] modifies the Ring-LWE key exchange so that it includes the Fujisaki-Okamoto transform for active security. However, as the Fujisaki-Okamoto transform only applies to encryption schemes, [i.24] needs to consider the encryption scheme induced by the key exchange. The present clause describes a simpler version of this based on the passively secure key transport from clause A.1.5 and using the refined Fujisaki-Okamoto transform from [i.117].

The initiator's key generation is the same is in the key transport.

1) The initiator generates a private key s_i and private noise term e_i, and computes the corresponding public key $B_i = s_i \cdot A + e_i$. The initiator sends their public key B_i to the responder.

The responder chooses a secret key K and encrypts it using a one-time pad. The seed for the one-time pad is then encrypted using the Ring-LWE key transport.

2) The responder chooses a random value $seed$ and encrypts the secret key K as $Z = KDF(seed) \oplus K$.

3) The responder uses $H(seed \parallel Z)$ to generate a private key s_r and private noise term e_r via a deterministic process, and computes the corresponding public key $B_r = s_r \cdot A + e_r$.

4) The responder uses $H(seed \parallel Z)$ to generate a second noise term e'_r via a deterministic process, and forms the value $C = s_r \cdot B_i + e'_r + \mathrm{Floor}(q/2) \cdot seed$. The responder sends their public key B_r, the blinded seed C and the encrypted secret key Z to the initiator.

The initiator recovers a putative seed from C and uses it to recreate the responder's side of the key transport.

5) The responder forms the value $W = C - s_i \cdot B_r$ and rounds $seed_f = \mathrm{Round}(2W/q) \bmod 2$.

6) The initiator regenerates values for the responder's private key s_f and private noise term e_f from $H(seed_f \parallel Z)$. The initiator then recomputes the responder's public key $B_f = s_f \cdot A + e_f$.

7) The initiator regenerates the responder's second noise term e'_f from $H(seed_f \parallel Z)$ and re-blinds the seed as $C_f = s_f \cdot B_i + e'_f + \mathrm{Floor}(q/2) \cdot seed_f$.

If the key transport was performed honestly then the initiator decrypts the session key.

8) The initiator checks that the recomputed public key B_f and value C_f match the public key B_r and value C that were sent by the responder. If they match then the initiator decrypts the secret key as $K = KDF(seed_f) \oplus Z$. Otherwise, the initiator treats it as a decryption failure.

NOTE: If the Fujisaki-Okamoto transform is applied to the Ring-LWE key exchange described in clause 6.3 then the noise term e''_r used by the responder for the randomized rounding during key extraction and check field calculation would also need to be deterministically generated from $H(seed_f \parallel Z)$.

Annex B:
SIDH background and security considerations

B.1 Mathematical background

B.1.1 Isogenies

The present annex provides a brief mathematical background on isogenies. Further details can be found in the references [i.45], [i.46] and [i.118].

Let E and E' be two elliptic curves over a finite field $\mathrm{GF}(q)$ of order q. An isogeny φ is a map from E to E' of the form:

$$\varphi(x, y) = \left(\frac{f_1(x, y)}{g_1(x, y)}, \frac{f_2(x, y)}{g_2(x, y)}\right),$$

for some polynomials $\mathrm{f}_1, \mathrm{f}_2, \mathrm{g}_1$ and g_2 in two variables, such that $\varphi(\infty)= \infty$ where ∞ denotes the identity element on the elliptic curve. Equivalently, an isogeny is a group homomorphism of the above form. In particular, given a point $aP + bQ$, where $P, Q \in E$ and $a, b \in \mathrm{GF}(q)$:

$$\varphi(aP + bQ) = \varphi(aP) + \varphi(bQ) = a\varphi(P) + b\varphi(Q) \in E'.$$

Two elliptic curves are said to be isogenous if there is an isogeny between them. The degree of the isogeny is its degree as a rational function. Given an isogeny $\varphi: E \longrightarrow E'$ of degree n, there exists another isogeny $\hat{\varphi}: E' \longrightarrow E$ also of degree n such that $\varphi \circ \hat{\varphi} = \hat{\varphi} \circ \varphi = [n]$. where $[n]$ is the map corresponding to multiplication by n. It follows that isogenies give an equivalence relation between elliptic curves. The isogeny $\hat{\varphi}$ is called the dual isogeny of φ.

For any natural number n, let $E[n]$ denote the n-torsion subgroup:

$$E[n] = \{P \in E(\overline{\mathrm{GF}(q)}) : nP = \infty\}.$$

In other words, $E[n]$ is the kernel of the multiplication by n map over the algebraic closure of $\mathrm{GF}(q)$. The group $E[n]$ is isomorphic to $(\mathbb{Z}/n\mathbb{Z})^2$ whenever n and q are relatively prime [i.118]. Accordingly, $E[n]$ has a basis consisting of two points, referred to as the n-torsion basis of E.

The endomorphism ring $\mathrm{End}(E)$ is defined to be the set of all isogenies from E to itself defined over the algebraic closure $\mathrm{GF}(q)$. The endomorphism ring is a ring under the addition of points and composition of functions. If the dimension of $\mathrm{End}(E)$ over $\mathbb{Z}$ is 2 then the elliptic curve is said to be ordinary; otherwise the dimension is 4 and the elliptic curve is said to be supersingular. Two isogenous curves are either both ordinary or both supersingular. All elliptic curves used in clause 7 are supersingular. One important property to note is that the endomorphism ring for supersingular elliptic curves is non-commutative.

Evaluating isogenies directly is inefficient. For efficient implementation, an isogeny can be represented by its kernel:

$$\ker(\varphi) = \left\{P \in E(\overline{\mathrm{GF}(q)}) : \varphi(P) = \infty\right\}$$

and its kernel can be used to evaluate it. Let $\langle X, Y, \ldots \rangle$ denote the group generated by $\{X, Y, \ldots\}$. For all of the isogenies considered in clause 7, the kernel will be generated by a single elliptic curve point R. Given an isogeny $\varphi: E \longrightarrow E'$ with kernel $\ker(\varphi) = \langle R \rangle$, the image curve E' can be expressed as $E/\langle R \rangle$. Consequently, the isogenies in clause 7 correspond directly to elliptic curve points. References [i.46] and [i.47] describe algorithms to evaluate isogenies given the corresponding kernel point.

B.1.2 Parameter generation

The public parameters of the SIDH key exchange are the prime p, the supersingular elliptic curve E, and the two pairs of auxiliary points $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$. These can be generated randomly by a trusted third party during the setup of the scheme.

This prime p has the form $p = l_A^{e_A}\, l_B^{e_B} f \pm 1$, where l_A and l_B are small primes and f is an optional cofactor. Typically, l_A and l_B are chosen to be the smallest prime values possible; that is, 2 and 3. The integers e_A, e_B are chosen so that $l_A^{e_A}$ and $l_B^{e_B}$ both have a bit length of approximately $n/2$ bits where n is the desired bit length for p. Different values for the cofactor f can then be tested until one is found for which either $p = l_A^{e_A}\, l_B^{e_B} f - 1$ or $p = l_A^{e_A}\, l_B^{e_B} f + 1$ is prime.

A method for constructing a specific supersingular curve E_0 over $\mathrm{GF}(p^2)$ is described in [i.119]. When $p \equiv 3 \pmod 4$, this gives the curve $y^2 = x^3 + 1$. A random supersingular curve E over $\mathrm{GF}(p^2)$ can be generated using a random walk on the isogeny graph starting from E_0.

To generate the auxiliary points $\{P_A, Q_A\}$, first randomly choose a point $P' \in E$ and compute $P = (l_B^{e_B} f)^2 P' \in E[l_A^{e_A}]$. If P has order $l_A^{e_A}$ then set $P_A = P$; otherwise, try again with a different P'. Next, randomly choose a point $Q' \in E$ and compute $Q = (l_B^{e_B} f)^2 Q' \in E[l_A^{e_A}]$. If Q has order $l_A^{e_A}$ then set $Q_A = Q$; otherwise, try again with a different Q'. Finally, if the result of the Weil pairing $e(P_A, Q_A)$ also has order $l_A^{e_A}$ then P_A and Q_A generate the torsion subgroup $E[l_A^{e_A}]$ so can be used as the auxiliary points; otherwise, try again with a different Q'.

The process for generating the auxiliary points $\{P_B, Q_B\}$ is analogous.

B.1.3 Public key compression

Let n be the bit length of the prime p. An elliptic curve of the form $y^2 = x^3 + ax + b$ over $\mathrm{GF}(p^2)$ is represented by a pair of values $a, b \in \mathrm{GF}(p^2)$ which take $4n$ bits. An elliptic curve point $P = (x, y)$ also corresponds to a pair of values in $\mathrm{GF}(p^2)$. However, there are only two possible values of y for each x-coordinate so it is possible to represent the point with only $2n + 1$ bits. Consequently, a public key can be represented using $8n + 2$ bits as it consists of a curve and two points.

In [i.120], the authors observe that for the SIDH key exchange either of the two possible y values for the points in the public key can be used in the computation of the shared secret key. They further suggest a method of constructing a canonical choice of curve from a j-invariant and propose representing points on the curve as linear combinations of a canonical basis. These techniques halve the size of a public key as it can now be represented by a j-invariant, which takes n bits; four coefficients, each of which takes $n/2$ bits; and a single-bit twist indicator. However, the reduction in the key size comes at a large computational cost.

A more recent work [i.53] presents an alternative implementation of key compression which has even smaller key sizes and is much faster computationally. By normalizing the two elliptic curve points that are part of the public key, it is possible to represent the pair with only $7n/2$ bits which makes the size of the public $7n/2 + 1$ bits.

B.2 Security

B.2.1 Provable security

The SIDH key exchange depends on the difficulty of certain isogeny problems. The most general form of the computational isogeny problem is the following: given a pair of values $j, j' \in \mathrm{GF}(q)$ find an isogeny $\varphi: E \to E'$, if exists, for elliptic curves E and E' over $\mathrm{GF}(q)$ where $j(E) = j$ and $j(E') = j$. However, the use of auxiliary points means that the isogeny problems underlying the SIDH key exchange are more specialized.

Let E be a supersingular elliptic curve with pairs of auxiliary points $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ as in clause 7.2.2. Further, let $\varphi_A: E \longrightarrow E_A$ be an isogeny of degree $l_A^{e_A}$. The computational supersingular isogeny problem is to find a generator R_A for the kernel of φ_A given the curve E_A and the images $\varphi_A(P_B), \varphi_A(Q_B)$ of the auxiliary points P_B, Q_B. Note that if the images $\varphi_A(P_A), \varphi_A(Q_A)$ are also known then a generator $R_A = m_A P_A + n_A Q_A$ can by recovered by finding m_A, n_A such that $m_A \varphi_A(P_A) + n_A\, \varphi_A(Q_A) = \infty$ via a discrete logarithm problem in a group of smooth order $l_A^{e_A}$.

The decisional version of this problem is to determine whether there exists an isogeny $\varphi_A: E \longrightarrow E_A$ of degree $l_A^{e_A}$ such that $\varphi_A(P_B) = {P'}_B$ and $\varphi_A(Q_B) = {Q'}_B$ given a curve E_A and a pair of points ${P'}_B, {Q'}_B \in E_A[l_B^{e_B}]$. An oracle for the decisional problem can be used to solve the computational problem by recovering the isogeny path in a polynomial number of steps [i.49].

Let $\varphi_A: E \longrightarrow E_A$ be an isogeny of degree $l_A^{e_A}$ with kernel generated by $R_A = m_A P_A + n_A Q_A$ and $\varphi_B: E \longrightarrow E_B$ be an isogeny of degree $l_B^{e_B}$ with kernel generated by $R_B = m_B P_B + n_B Q_B$. The computational supersingular Diffie-Hellman problem is to find the j-invariant of the curve $E/\langle R_A, R_B\rangle \cong E_A/\langle\varphi_A(R_B)\rangle \cong E_B/\langle\varphi_B(R_A)\rangle$ given the curves E_A, E_B and the images $\varphi_A(P_B), \varphi_A(Q_B) \in E_A[l_B^{e_B}]$ and $\varphi_B(P_A), \varphi_B(Q_A) \in E_B[l_A^{e_A}]$.

References [i.46] and [i.49] contain further discussion of these assumptions.

B.2.2 Passive security

As described in clause B.2.1, the general hard problem is to find an isogeny between two elliptic curves that are known to be isogenous. When the curves are ordinary, their endomorphism ring is commutative and so the quantum algorithm for finding a hidden shift subgroup can be used to give a subexponential attack [i.121]. For supersingular curves, the endomorphism ring is no longer commutative and the hidden shift subgroup problem no longer applies. The fastest known algorithm to recover an isogeny between supersingular curves over $\mathrm{GF}(p^2)$ is a quantum meet-in-the-middle attack which has complexity $O(\sqrt[6]{p})$. Similarly, the fastest known classical algorithm to recover a supersingular isogeny is a classical meet-in-the-middle attack which has complexity $O(\sqrt[4]{p})$. Consequently, an n-bit prime p provides approximately $n/6$ bits of quantum security and $n/4$ bits of classical security [i.46].

The choice of the elliptic curve E in the public parameters does not affect the security of the scheme because isogenies between elliptic curves form a Ramanujan expander graph [i.46], so from any starting curve it is possible to move to a random curve in the isogeny graph with roughly equal probability using only a small number of isogeny steps. There are more efficient quantum [i.122] and classical [i.123] algorithms for constructing isogenies between supersingular curves defined over $\mathrm{GF}(p)$. These form a negligible fraction of the curves in the isogeny graph so the algorithms do not pose a threat to SIDH.

The choice of auxiliary points does not affect the security of the scheme because it is straightforward to switch between any pair of generators for the torsion subgroup using a simple linear transformation. A recent paper by Petit [i.124] does describe a polynomial-time algorithm to reconstruct an isogeny between supersingular curves over $\mathrm{GF}(p^2)$ using the images of the auxiliary points under the isogeny. However, the algorithm requires one of the torsion subgroups to be significantly larger than the other. Consequently, it does not apply when the parameters are chosen so that the two torsion subgroups are approximately the same size, as in clause 7.2.2.

B.2.3 Active security

Galbraith et al [i.48] suggest using the variant of the Fujisaki-Okamoto transform described in [i.50] to provide active security. This applies directly to the key exchange without first needing to convert it into an encryption scheme, but it does not have a proof of security.

The initiator's key generation is the same as in the original SIDH key exchange.

1) The initiator generates a point $R_A \in \langle P_A, Q_A\rangle$, uses this to compute a private isogeny $\varphi_A: E \to E_A = E/\langle R_A\rangle$ and then calculates the images $\varphi_A(P_B)$ and $\varphi_A(Q_B)$. The initiator sends the curve E_A and images $\varphi_A(P_B)$, $\varphi_A(Q_B)$ to the responder.

The responder generates their key pair from a random seed. The seed is then encrypted using the shared value derived from the key exchange.

2) The responder chooses a random value $seed$ and uses this to generate a point $R_B \in \langle P_B, Q_B\rangle$ via a deterministic process. The responder computes a private isogeny $\varphi_B: E \to E_B = E/\langle R_B\rangle$ and then calculates the images $\varphi_B(P_A)$ and $\varphi_B(Q_A)$.

3) The responder uses the initiator's public points $\varphi_A(P_B)$, $\varphi_A(Q_B)$ to recover the image $\varphi_A(R_B)$ of R_B under the initiator's private isogeny φ_A. The responder computes the curve $E_{BA} = E_A/\langle\varphi_A(R_B)\rangle$ and derives the shared value S from the j-invariant of E_{BA}.

4) The responder derives a session key $K = KDF_1(S)$ from the shared value S.

5) The responder encrypts the random value as $Z = KDF_2(S) \oplus seed$. The responder sends their curve E_B, images $\varphi_B(P_A)$, $\varphi_B(Q_A)$, and encrypted seed Z to the initiator.

The initiator completes the key exchange to recover a putative seed and uses it to recreate the responder's side of the key transport.

6) The initiator uses the responder's public points $\varphi_B(P_A)$ and $\varphi_B(Q_A)$ to recover the image $\varphi_B(R_A)$ of R_A under the responder's private isogeny φ_B. The initiator computes the curve $E_{AB} = E_B/\langle\varphi_B(R_A)\rangle$ and derives the shared value S' key from the j-invariant of E_{AB}.

7) The initiator recovers $seed' = KDF_2(S') \oplus Z$ and uses this to regenerate the responder's point R'_B, isogeny $\varphi'_B: E \to E'_B = E/\langle R'_B\rangle$, and images $\varphi'_B(P_A)$ and $\varphi'_B(Q_A)$.

If the key exchange was performed honestly then the initiator derives the session key.

8) The initiator checks that the recomputed curve E'_B and images $\varphi'_B(P_A)$, $\varphi'_B(Q_A)$ match the curve E_B and images $\varphi_B(P_A)$, $\varphi_B(Q_A)$ sent by the responder. If they match then the initiator derives the session key $K = KDF_1(S')$. Otherwise, the initiator treats it as a key exchange failure.

Annex C:
Open Quantum-Safe benchmarks

C.1 Open Quantum-Safe

The Open Quantum-Safe project [i.125] is developing a library of quantum-safe primitives together with a common testing and benchmarking framework. The primitives currently integrated into the library include examples of the key exchanges considered in clauses 6, 7 and 8:

- Ring-LWE key exchanges (BCNS [i.28] and New Hope [i.27] and [i.126]);
- LWE key exchange (Frodo [i.25]);
- Isogeny-based key exchange (Jao-De Feo [i.46] and SIDH [i.47]);
- Niederreiter key transport (McBits [i.66]); and
- NTRU key transport.

The present annex compares the performance of the various key exchanges using benchmarks produced by the Open Quantum-Safe library on three different hardware platforms:

- Desktop with a 64-bit AMD A10-6700 quad-core processor operating at 3,7 GHz;
- Laptop with a 64-bit Intel® Pentium® T4400 dual-core processor operating at 2,2 GHz; and
- Single board computer with a 32-bit ARM1176™ core operating at 700 MHz.

The performance estimates give the median time to perform the initiator's key generation; the responder's key generation or encryption; and the initiator's reconciliation or decryption. The benchmarks were taken when the processors were idle except for standard operating system background processes.

C.2 Benchmarks

C.2.1 Performance on a 64-bit desktop processor

Table C.1 gives performance estimates (in milliseconds) for the key exchanges on a 64-bit AMD A10-6700 quad-core desktop processor operating at 3,7 GHz and with Turbo Core disabled. The host operating system was Microsoft® Windows® 8.1, but the library was built using gcc version 5.4.0 in an Ubuntu® 16.04 virtual machine managed by Oracle® VirtualBox™ 5.0.32. The compiler optimization flag was set to -O2 and the library was configured to allow the use of AVX and AES-NI instructions. AVX2 instructions are not supported by the A10-6700 processor.

Table C.1: Performance on a 64-bit desktop processor

Scheme	Performance:			Notes
	Initiator (Start)	Responder	Initiator (End)	
New Hope	0,092 ms	0,165 ms	0,034 ms	[i.126]
Frodo	5,95 ms	6,05 ms	0,137 ms	[i.25]
SIDH	22,5 ms	50,3 ms	21,3 ms	[i.47]
McBits	206 ms	0,083 ms	0,204 ms	[i.66]
NTRU	2,00 ms	0,282 ms	0,143 ms	[i.78]

C.2.2 Performance on a 64-bit laptop processor

Table C.2 gives performance estimates (in milliseconds) for the key exchanges on a 64-bit Intel® Pentium® T4400 dual-core laptop processor operating at 2,2 GHz. The operating system was Ubuntu® 16.04 and the library was built using gcc version 5.4.0. The compiler optimization flag was set to -O2 and the library was configured to allow the use of AVX instructions. AVX2 and AES-NI instructions are not supported by the Pentium® T4400 processor.

Table C.2: Performance on a 64-bit laptop processor

Scheme	Performance:			Notes
	Initiator (Start)	Responder	Initiator (End)	
New Hope	0,134 ms	0,237 ms	0,044 ms	[i.126]
Frodo	69,8 ms	71,5 ms	0,232 ms	[i.25]
SIDH	35,1 ms	78,3 ms	33,2 ms	[i.47]
McBits	275 ms	0,136 ms	0,299 ms	[i.66]
NTRU	3,04 ms	0,383 ms	0,244 ms	[i.78]

C.2.3 Performance on a 32-bit embedded processor

Table C.3 gives performance estimates (in milliseconds) for the key exchanges on a single board computer containing a 32-bit ARM1176™ core operating at 700 MHz. The operating system was Debian® 7.11 and the library was built using gcc version 4.6.3. The compiler optimization flag was set to -O2. The AVX, AVX2 and AES-NI instructions are not supported by the ARM1176™ core.

Table C.3: Performance on a 32-bit embedded processor

Scheme	Performance:			Notes
	Initiator (Start)	Responder	Initiator (End)	
New Hope	1,60 ms	2,68 ms	0,411 ms	[i.126]
Frodo	858 ms	880 ms	2,12 ms	[i.25]
SIDH	3 270 ms	7 340 ms	3 090 ms	[i.47]
McBits	10 300 ms	2,15 ms	6,32 ms	[i.66]
NTRU	33,2 ms	5,14 ms	2,49 ms	[i.78]

C.3 Discussion

It is clear from tables C.2 and C.3 that removing support for AVX instructions significantly reduces the performance of SIDH and McBits key generation. New Hope, Frodo and NTRU are all approximately 10 times slower on the 32-bit ARM core than on the 64-bit Intel® processor whereas McBits key generation is almost 40 times slower and SIDH is nearly 100 times slower.

Similarly, tables C.1 and C.2 show a significant drop in the performance of key generation for Frodo when AES-NI instructions are not supported. Although the performance of the other algorithms on the 64-bit Intel® processor is close to their performance on the 64-bit AMD processor, key generation for Frodo is over 10 times slower. AES is only used during the pseudo-random generation of the public matrix A and if this is replaced by a fixed public matrix then the performance of Frodo is not affected to the same extent.

History

Document history		
V1.1.1	October 2017	Publication