HOUGAARD, HECTOR BJOLJAHN

STUDY ON POST-QUANTUM GROUP KEY EXCHANGES AND GENERALIZATION

STUDY ON POST-QUANTUM GROUP KEY EXCHANGES AND GENERALIZATION

HOUGAARD, HECTOR BJOLJAHN

A Ph.D. Dissertation

Graduate School of Engineering
Osaka University

Supervisor: Prof. Atsuko Miyaji

January 2022

No man ever steps in the same river twice, for it's not the same river
and he's not the same man.

— Heraclitus (544 BC - 483 BC)

Dedicated to my family.
Without whose support I would not be here.

ABSTRACT

The advent of quantum computing has seen many classically secure cryptographic constructions become potentially insecure in the near future. Quantum computers can easily solve problems that most current cryptographic constructions use as building blocks, namely the integer factorization problem, the discrete logarithm problem, and the elliptic curve discrete logarithm problem. This means that we will need to discard those cryptographic constructions or risk having information fall into malicious hands. The nature of cryptographic proofs offers us an alternative: Fix them. By generalizing the constructions and updating the basis for their security, we can preserve these constructions while guarding against any future adversaries, quantum included. So far, quantum-secure, or post-quantum cryptography (PQC) secure, key exchanges, signature schemes, encryption schemes, and more have been proposed but there are still classical constructions that have no PQC alternative and there are many improvements that can be made in security, key size, and speed (complexity).

In this dissertation, we consider the generalization of classical constructions to PQC secure versions and beyond. In particular, we consider group key exchanges (GKEs) and pseudorandom permutations (PRPs). PQC secure GKEs do exist but they all have linear order complexity so an improvement would be welcome. PQC secure PRPs have yet to be constructed but recent PQC secure pseudorandom functions (PRFs) point towards their inevitability.

In Chp. 3, we show that it is possible to define a GKE using the isogeny based PQC primitive, supersingular isogeny Diffie-Hellman (SIDH), to create a logarithmic order complexity GKE. We further give a peer-to-peer/sequential version with similar complexity. We also show that there is a compiler that turns both GKEs into authenticated group key exchanges (AGKEs), also with logarithmic order complexity.

In Chp. 4, we show that it is also possible to define a GKE using the lattice based PQC primitive, ring-learning-with-errors (R-LWE), to create a logarithmic order complexity GKE. We also give a peer-to-peer/sequential version and show that both can be compiled into logarithmic order complexity AGKEs.

In Chp. 5, we take advantage of the structure of cryptography together with the GKEs in Chp. 3 and Chp. 4, generalize the definitions of key exchanges and hard problems, and then give generic compilers for group key exchange, both a concurrent and sequential version.

In Chp. 6, we generalize the Even and Mansour block cipher construction to using group actions as their underlying operations, giv-

ing us a PRP that is secure against an unbounded adversary having polynomially-many classical oracle queries.

Finally, we conclude by summarizing our results, discussing their relation to the wider field of cryptography, and considering future work based on them.

LIST OF PUBLICATIONS

Journals

1. Hougaard, H. B. and Miyaji, A. **Authenticated logarithmic-order supersingular isogeny group key exchange**. International Journal of Information Security. Springer, 2021.
   https://doi.org/10.1007/s10207-021-00549-4

2. Hougaard, H. B. and Miyaji, A. **Authenticated tree-based R-LWE group key exchange**. The Computer Journal. Oxford Press, 2021.
   https://doi.org/10.109/comjnl/bxab165

Conferences

1. Hougaard, H. B. and Miyaji, A. **SIT: supersingular isogeny tree-based group key exchange**. 2020 15th Asia Joint Conference on Information Security (AsiaJCIS), pp. 46-53. IEEE Computer Society, 2020.
   https://doi.org/10.1109/AsiaJCIS50894.2020.00019

2. Hougaard, H. B. and Miyaji, A. **Tree-based ring-LWE group key exchanges with logarithmic complexity**. In: Meng W., Gollmann D., Jensen C.D., Zhou J. (eds) Information and Communications Security. ICICS 2020. LNCS, vol 12282, pp. 91-106. Springer, 2020.
   https://doi.org/10.1007/978-3-030-61078-4_6

3. Hougaard, H. B. and Miyaji, A. **Group key exchange compilers from generic key exchanges**. International Conference on Network and System Security (NSS). LNCS. Springer, 2021.
   To be published

Conference and workshop presentations

1. Hougaard, H. B. and Cheng, C. **Generic Even-Mansour construction based on group actions**. Indo-Japan 2020 student presentation. Department of Science and Technology (DST). Calcutta, India. January 2020.

2. Hougaard, H. B. and Cheng, C. and Miyaji, A. **Generic Even-Mansour construction based on group actions**. IEICE Tech. Rep., vol. 119, no. 140, ISEC2019-37, pp. 215-220, 2019.

ACKNOWLEDGMENTS

Many thanks to my advisor, Prof. Atsuko Miyaji, for giving me the freedom to research what I wanted to and all the great feedback she gave me on my articles. Thanks to Assoc. Prof. Chen-Mou Cheng for the conversations about the generalizing nature of cryptography that became the theme of this dissertation.

As to those invaluable persons who reviewed my dissertation and thereby made it better, I give my thanks, first and foremost, to my external reviewer, Dr. Mehdi Tibouchi of Okamoto Research Laboratory at the NTT Secure Platform Laboratories (Tokyo) and affiliated with the Graduate School of Informatics at Kyoto University, and my internal reviewers, Prof. Noboru Babaguchi and Prof. Tetsuya Takine from the Division of Information and Communications Technology, in the Department of Electrical, Electronic, and Information Engineering, at the Graduate School of Engineering, Osaka University. I also give my thanks to the rest of the examination committee: Prof. Kyo Inoue, Prof. Kazunori Komatani, Prof. Akihiro Maruta, Prof. Seiichi Sampei, and Prof. Takashi Washio, also from the Division of Information and Communications Technology, in the Department of Electrical, Electronic, and Information Engineering, at the Graduate School of Engineering, Osaka University.

I give my humblest thanks to the Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT) for giving me the chance to study in Japan for the duration of my Ph.D.. I thank them for the support, financial and otherwise, that the Monbukagakusho (MEXT) scholarship afforded me. It has been an experience that has changed me greatly and I will be grateful for it for the rest of my life.

Thanks to everyone that gave me support throughout my Ph.D. time, especially when I needed it.

CONTENTS

ABSTRACT vii

LIST OF PUBLICATIONS ix

ACKNOWLEDGEMENTS xi

I BACKGROUND

1 INTRODUCTION 3
- 1.1 Motivation 3
- 1.2 Post-quantum cryptography 4
- 1.3 Group key exchanges 5
- 1.4 Pseudorandom permutations 7
- 1.5 Contributions and Copyright Notices 8
- 1.6 Outline 9

2 PRELIMINARIES 11
- 2.1 Notation 11
- 2.2 Algebra 11
- 2.3 Cryptography 15
  - 2.3.1 Diffie-Hellman 18
  - 2.3.2 Signature schemes 18
  - 2.3.3 Pseudorandomness 19
- 2.4 Burmester-Desmedt group key exchanges 21
- 2.5 Supersingular isogeny Diffie-Hellman 23
- 2.6 Ring-learning-with-errors (R-LWE) 26
- 2.7 Even-Mansour 29
- 2.8 Security models 30
  - 2.8.1 Security model 30
  - 2.8.2 G-CK$^+$ security model 33
  - 2.8.3 Security model discussion 34

II RESEARCH

3 GROUP KEY EXCHANGES FROM ISOGENIES 37
- 3.1 Introduction 37
- 3.2 Related work 39
- 3.3 Supersingular isogeny tree-based GKE (SIT) 40
  - 3.3.1 CSIDH version 45
- 3.4 Peer-to-peer SIT (P2P-SIT) 46
- 3.5 Authenticated SIT (A-SIT) 48
- 3.6 Comparison 52
- 3.7 Concluding remarks 55

4 GROUP KEY EXCHANGES FROM RING-LEARNING-WITH-ERRORS 57
- 4.1 Introduction 57
- 4.2 Related work 57

4.3 R-LWE tree-based GKE (Tree-R-LWE-GKE) 58
4.4 Peer-to-peer R-LWE group key exchange (P2P-Tree-R-LWE-GKE) 62
4.5 Authenticated Tree-R-LWE-GKE 63
4.6 Comparison 68
4.7 Concluding remarks 69
5 GENERIC GROUP KEY EXCHANGE COMPILERS 71
5.1 Introduction 71
5.2 Preliminaries 71
5.3 Generalized group key exchange compiler (GKE-C) 74
5.4 Peer-to-peer GKE-C (P2P-GKE-C) 77
5.5 Complexity analysis 80
5.6 Conclusion 81
6 GENERALIZED EVEN-MANSOUR 83
6.1 Introduction 83
6.2 Preliminaries 84
6.3 Related work 84
6.4 Group action Even-Mansour 84
6.4.1 Evidence for quantum security 92
6.5 Concluding remarks 93

III CONCLUSION

7 CONCLUDING REMARKS 97
7.1 Summary 97
7.2 Discussion 97
7.3 Future work 98

IV APPENDIX

A TABLE OF KEY EXCHANGES IN THE 0/1-IKE NOTATION AND THEIR CORRESPONDING HARD PROBLEMS 103

BIBLIOGRAPHY 105

LIST OF FIGURES

Figure 1.1 Overview of our results 9
Figure 2.1 Reduction proof overview [40] 17
Figure 2.2 BDI circle structure 21
Figure 2.3 BDII tree-based structure 22
Figure 2.4 SIDH key exchange 25
Figure 3.1 Possible SIT graph configuration for $n = 12$ 41
Figure 3.2 The neighbours of $\mathcal{P}_i$: Parent and children 41
Figure 6.1 Game R and Game X 87
Figure 6.2 Game X′ 88
Figure 6.3 Game R′ 90

LIST OF TABLES

Table 2.1 Exclusive-OR (XOR) operation. 12
Table 3.1 Detailed comparison table of isogeny-based GKEs, assuming a binary double-tree for SIT and P2P-SIT. **I** and **S** denote isogeny tuples and summed values, respectively. Values in square brackets are particular for leaf nodes, if they differ from other nodes. 53
Table 3.2 Comparision overview of isogeny-based GKEs and AGKEs, assuming a balanced double-tree for SIT, P2P-SIT and their authenticated versions, with l children per non-leaf node. 53
Table 4.1 Comparision overview of R-LWE based AGKEs. 68
Table 5.1 Examples of GKE-C compiled GKEs. 80
Table 5.2 Comparison of our GKE compilers. $\mathfrak{A}$ denotes a value from the underlying 0/1-IKE while $\mathfrak{G}$ denotes a value from the GKE-C. 81

AGKE authenticated group key exchange

AKE authenticated key exchange

A-P2P-SIT authenticated P2P-SIT

A-SIT authenticated SIT

BDI Burmester-Desmedt I

BDII Burmester-Desmedt II

CSIDH commutative supersingular isogeny Diffie-Hellman key exchange

CSSI computational supersingular isogeny

DES Data Encryption Standard

DH Diffie-Hellman key exchange

DDH decisional Diffie-Hellman

DDH-like decision Diffie-Hellman-like

DLP discrete logarithm problem

D-R-LWE decisional ring-learning-with-errors

DSSI decisional supersingular isogeny

ECC elliptic curve cryptography

ECDLP elliptic curve discrete logarithm problem

EM Even-Mansour

GAEM group action Even-Mansour

G-CK$^+$ group-Canetti-Krawczyk plus

GDH generic decisional hard

GKE group key exchange

GKE-C GKE compiler

IKE interactive KE

Isog-GKE isogeny-based GKE

Isog-AGKE isogeny-based AGKE

KE key exchange

NIST National Institute of Standards and Technology

P2P-GKE-C peer-to-peer GKE compiler

P2P-SIT peer-to-peer SIT

P2P-Tree-R-LWE-GKE peer-to-peer tree-based R-LWE GKE

PKI public key infrastructure

PPT probabilistic polynomial-time

PQC post-quantum cryptography

PRF pseudorandom function

PRP pseudorandom permutation

R-LWE ring-learning-with-errors

R-LWE KE R-LWE key exchange

SIBD supersingular isogeny Burmester-Desmedt

SIDH supersingular isogeny Diffie-Hellman

SIT supersingular isogeny tree-based GKE

SPRP super pseudorandom permutation

SSCDH supersingular computational Diffie-Hellman

SSDDH supersingular decisional Diffie-Hellman

Tree-R-LWE-GKE tree-based R-LWE GKE

XOR exclusive-OR

Part I

BACKGROUND

1 INTRODUCTION

1.1 MOTIVATION

Quantum computing is an exciting field full of potential for improving the lives of humans everywhere. However, with this potential for benefits comes also the risk of abuse. The reason is that quantum computers have the potential to break most current cryptography, leading to security concerns for personal, public, and governmental information.

At present, post-quantum cryptography (PQC) is an attempt to safeguard current cryptography against these future quantum adversaries as sensitive, encrypted data can be stockpiled and decrypted later. The US National Institute of Standards and Technology (NIST), the leading body pressing for a standardization of PQC protocols, initiated the NIST competition in 2017 for exactly this reason [14]. The competition has seen several PQC candidates proposed for encryption schemes, key exchanges, and signature schemes, i.e. cryptographic primitives, and is currently in its third round, having discarded dozens of potential candidates. Among its selection criteria are security, key size, and speed. Although, the NIST competition is small in scope, the impact has been large. Already there are several PQC protocols proposed outside of the NIST competition based on these primitives: Fundamental protocols such as group key exchanges and multi-signatures to blockchains and zero-knowledge proofs, not to mention speed-ups of, and attacks on, many NIST candidates themselves. From the interest within the cryptography community - PQCrypto already having become a major cryptography conference since its start in 2006, annual from 2016 - it is clear that there is an interest in PQC constructions. From the inevitability of the quantum menace, the industry too will soon be forced to adopt PQC as its standard.

What this dissertation aims to do, is to consider whether previous cryptographic constructions, those that would be broken by quantum adversaries, could be made secure by changing the underlying primitives to PQC candidates. This requires that the constructions can be generalized to structures that permit the PQC candidates to work and that their security can be proven as well.

Building a library of constructions that are quantum-secure is of course the priority at the moment, especially for basic cryptographic constructions, but there is also a bigger question looming: What happens when an even faster and stronger adversary appears? We believe that the structure of cryptography, the reliance on proofs by reduction,

is the saving grace of information security. If we are able to generalize the structure of cryptography itself, the definitions and constructions, then it would be possible to create an entire field of cryptographic constructions based on even a single building block. Hence, this is our ultimate goal and the theme of this dissertation, although only a beginning towards such an endeavour.

1.2 POST-QUANTUM CRYPTOGRAPHY

Post-quantum cryptography (PQC) deals with using classical computers for cryptography while defending against quantum computers. Four major branches of mathematics/information theory for use in PQC have already been identified: Lattice-, multivariate-, code-, and isogeny-based.

Each branch relies on their own collection of 'hard' problems, i.e. mathematical or computational problems that are infeasible to solve using known algorithms, both classical and quantum. The cryptographic constructions such as key exchanges, signature schemes, and encryption schemes that are based on these hard problems arise from practical needs.

The two PQC candidates used in this dissertation are supersingular isogeny Diffie-Hellman (SIDH) [38] and R-LWE key exchange (R-LWE KE) [20, 52]. SIDH is based on isogenies between elliptic curves. Elliptic curves are a special type of curve that have long been studied in mathematics and used in cryptography, and isogenies are essentially functions between elliptic curves. The security of this primitive comes from the seemingly difficult task of finding an isogeny between two given elliptic curves. R-LWE KE is a lattice-based key exchange, more specifically based on ring-learning-with-errors (R-LWE), a primitive that relies on error-compounding and the difficulty of error-correcting said compounding.

Each PQC candidate has its pros and cons. For example, SIDH is rather slow to compute but has a small public key size per party, approximately 18 milliseconds on a 5 GHz computer with a 564 byte (can be compressed to 330 bytes) public key size for 128-bit quantum security [15], while R-LWE is fast to compute but has a larger public key size per party, approximately 2.1 milliseconds on a 5 GHz computer with a 960 byte public key size per party for 128-bit quantum security [7, 58]. For some uses, this can have a major impact as cryptographic implementations can have limited computation power or strict space requirements. For example, Tor - popularly used as the gateway to the deep web - uses data cells that require data be less than 517 bytes in length. Hence, the different strengths and weaknesses PQC candidates have must be considered for implementation purposes.

1.3 GROUP KEY EXCHANGES

Most cryptography relies on the establishment of a secure channel of communication over an otherwise insecure channel, i.e. only having the information be accessible to the intended recipient and no others, despite communicating over public channels where anyone can monitor the communication (and possibly even manipulate it). To establish a secure channel of communication, users employ a common secret key in order to "lock" sensitive information such that it can only be "unlocked" by a legitimate party. Without such a key, the entire communication protocol would be insecure.

At the advent of modern public-key cryptography, a protocol was invented that could establish a shared key between two parties, without having to meet physically, a so-called two-party, public-key key exchange protocol. Research into key exchanges progressed from there with a focus on security and on efficiency, both for time and resources. More security, for obvious reasons, and better efficiency because some information is time-sensitive or large in terms of memory storage (think gigabytes of data, back in the 1990s) and computers have a limit on their computational resources (though it is constantly improving; see for example Moore's law[1]). Originally, cryptography was used by governments for military and top-level secret communication such as the secure phone-line between Russia and the USA, and by large corporations, for example to secure business secrets. These bodies had large computational resources at their disposal, so efficiency was less important than security, but eventually cryptography was employed for use by consumers, especially on the internet.

As consumer-level computational power can be severely limited, efficiency has become a major factor. The uses for cryptography have also changed and the number of parties involved in a single cryptographic protocol have changed too. For example, IoT devices in a home that have a single key shared by all parties that have access to it or a group chat in the LINE or WhatsApp communication applications. Due to the complexity of the interactions between the parties involved, sharing a key can be a complicated and computationally intensive procedure. For a group of parties wishing to communicate securely, a two-party key exchange can be repeated for each pair of parties, but this quickly leads to very large numbers of keys being stored. For example, imagine that a group of 256 parties want to create a WhatsApp group.[2] Naively, each pair of parties would have to share keys in order to make sure they could see each message that the other parties might send. This would require the WhatsApp server to process $256 \cdot (256 - 1)/2 = 32640$ two-party key exchanges and

1 Moore's law loosely states that computing power will double approximately every two years.

2 Maximum number of parties in a WhatsApp group. [12]

each party would have to store 256 64-byte shared keys. This is, in fact,[3] what WhatsApp does and, granted, is not a large amount of information to store, but it could be severely reduced. This is because it is possible to use a group key exchange (GKE) where each party ends up with a single session key shared by all parties in the group where the number of computations, the information stored, and the time required are minimized. In the WhatsApp example, the key exchanges are redone every time a party leaves the group, which can consume the bandwidth of parties with low data-allowance if they engage in many groups.

If there is a need for verifying that no party outside a group has access to a new shared key, for example a party that had previously left the group, it is also possible to consider authenticated group key exchanges (AGKEs), where parties cannot be impersonated. For WhatsApp, the authentication is done by signing each message with a digital signature, a way to prove that the message was indeed sent by the sender and not an imposter.

Depending on the efflux and influx of group members, the staggering number of messages to be encrypted, signed, decrypted, authenticated, etc., might cause efficiency problems that could be alleviated by employing a GKE or AGKE protocol instead.

In the above Whatsapp example, the key size was rather small, only 64 bytes per party. This is because WhatsApp employs elliptic curve cryptography (ECC), which has small key sizes and is also computationally efficient (fast to compute) [5]. However, ECC is based on the elliptic curve discrete logarithm problem (ECDLP) and thereby insecure against quantum adversaries [54]. Even switching to a GKE based on one of the classical primitives would not help this situation. Instead, they could use one of the several PQC (A)GKEs that have been proposed.

In this dissertation, we choose to compare GKEs by their round, communication, and memory complexity. The round complexity is the maximum number of times any party must wait for information from other parties in order to proceed. The communication complexity considers the maximum number of messages *received* by any party in one call of the protocol. The memory complexity takes into account the maximum number of values stored until the session key computation. These complexities are dependent on the number of parties and can be expressed as functions thereof. Intuitively and factually, the slower this function grows, the better the complexity.

3 A new group member generates a 32-byte "Chain Key" and a random Curve25519 "Signature Key" key pair, then combines the Chain Key and 32-byte public key from the Signature Key into a "Sender Key" message. The Sender Key message is then encrypted using end-to-end encryption for each group member and sent individually. [64]

Fujioka et al. [26],[4] Furukawa et al [27], and Azarderakhsh et al. [4] all proposed isogeny based (A)GKEs while Apon et al. [3] and Choi et al. [13] proposed lattice based (A)GKEs. The best of these use only a few rounds of message exchanges and end up with linear communication and memory complexity. There also exists a three-round AGKE compiler from PQC primitives by Persischetti et al. [53] and it has linear order complexity, just like the AGKE compiler by Katz and Yung [41], the standard AGKE compiler in the literature. As such, the GKEs and compilers could be improved as there are two orders of magnitude below the linear complexity of the communication and memory complexities, namely constant and logarithmic.

The problems that we seek to solve are therefore to minimize the round, communication, and memory complexities while maintaining the same or an improved level of security. We manage to reduce the communication and memory complexities by an entire order of magnitude (linear to logarithmic) while the other properties are similar to PQC (A)GKEs based on the same primitives. The structure of our (A)GKE constructions also gives us highly flexible (A)GKEs that can be changed according to the needs of the network and individual users. Furthermore, we generalize the GKE structure such that it can be used as a compiler for other two-party, public-key key exchanges.

1.4 PSEUDORANDOM PERMUTATIONS

Pseudorandom permutations (PRPs) are a way to shuffle information in a way that is indistinguishable from random. The Even-Mansour (EM) block cipher [24] is a Data Encryption Standard (DES) [51] inspired block cipher, i.e. an encryption scheme that uses a symmetric key along with a deterministic algorithm in order to encrypt a message. However, Kilian and Rogaway [42] showed that the construction actually satisfies the property of being a PRP. The EM construction itself uses a PRP oracle in its algorithm however, so it is not an explicit PRP construction, but rather shows how to extend the usability of a PRP.

Pseudorandom functions (PRFs) satisfy a weaker notion of security as they are not expected to be invertible. Classically, we can even build PRPs from PRFs using a Feistel construction as shown by Luby and Rackoff [46]. However, a quantum adversary using quantum oracle queries is able to completely break the indistinguishability of this PRP construction as first shown by Kuwakado and Morii [43]. Zhandry [65] did however give some evidence that quantum-secure PRPs were theoretically possible and while no PQC secure PRP seems to have been created at present, Alamati et al. [2], Moriya et al. [50], and Boneh et al. [6] independently show the existence of group action based PQC PRFs. Alagic and Russell [1], on the other hand, gave some evidence that the EM construction might be a PQC secure construction when the

4 [26] assumes the existence of cryptographic invariant maps, a yet unproven claim.

underlying group (the group of bit strings) is exchanged with certain other groups. In Ch. 6, we therefore define and show classical security for the most general group-theoretic EM construction we can, in the hope that the construction may still be proven quantum secure.

1.5 CONTRIBUTIONS AND COPYRIGHT NOTICES

In this dissertation, we consider whether we can generalize classical schemes while improving the complexity and security of the schemes. We do so by looking at GKEs, AGKEs, and the EM scheme.

In Chp. 3, we show that it is possible to define a GKE using the isogeny based PQC primitive, SIDH, to create a logarithmic order complexity GKE. We further give a peer-to-peer/sequential version with similar complexity. We also show that there is a compiler that turns both GKEs into AGKEs, also with logarithmic order complexity. This chapter includes substantial portions of the conference article first published in

- Hector B. Hougaard and Atsuko Miyaji. "SIT: supersingular isogeny tree-based group key exchange." In: *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*. IEEE, 2020, pp. 46–53. DOI: 10.1109/AsiaJCIS50894.2020.00019

Reproduced with permission from ©2020 IEEE. This chapter also includes substantial portions of the journal article first published in

- Hector B. Hougaard and Atsuko Miyaji. "Authenticated logarithmic-order supersingular isogeny group key exchange." In: *International Journal of Information Security*. Springer, 2021. DOI: 10.1007/s10207-021-00549-4

Reproduced with permission from Springer Nature.

In Chp. 4, we show that it is also possible to define a GKE using the lattice based PQC primitive, ring-learning-with-errors (R-LWE), to create a logarithmic order complexity GKE. We also give a peer-to-peer/sequential version and show that both can be compiled into logarithmic order complexity AGKEs. This chapter includes substantial portions of the conference article first published in

- Hector B. Hougaard and Atsuko Miyaji. "Tree-Based Ring-LWE Group Key Exchanges with Logarithmic Complexity." In: *Information and Communications Security*. Ed. by W. Meng et al. Vol. 12282. LNCS. Springer International Publishing, 2020, pp. 91–106. DOI: 10.1007/978-3-030-61078-4_6

Reproduced with permission from Springer Nature. This chapter also includes substantial portions of the journal article first published in

- Hector B. Hougaard and Atsuko Miyaji. "Authenticated tree-based R-LWE group key exchange." In: *The Computer Journal*. Ed. by Oxford Press. 2021. DOI: 10.109/comjnl/bxab165

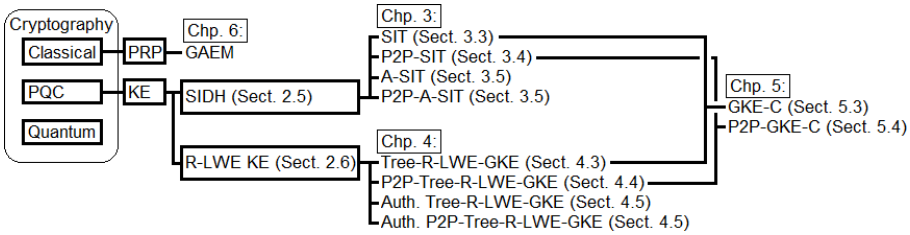

Figure 1.1: Overview of our results

Reproduced with permission from Oxford University Press.

In Chp. 5, we take advantage of the structure of cryptography together with the GKEs in Chp. 3 and Chp. 4, generalize the definitions of key exchanges and hard problems, and then give generic compilers for group key exchange, both a concurrent and sequential version. These GKE compilers do not only apply to PQC but to classical cryptography and future cryptographic paradigms as well. This chapter includes substantial portions of the conference article first published in

- Hector B. Hougaard and Atsuko Miyaji. "Group key exchange compilers from generic key exchanges." In: *International Conference on Network and System Security (NSS)*. LNCS. Springer International Publishing, 2021. DOI: 10.1007/978-3-030-92708-0_10

Reproduced with permission from Springer Nature.

In Chp. 6, we generalize the Even and Mansour block cipher construction to using group actions as their underlying operations, giving us a PRP that is secure against an unbounded adversary having polynomially-many classical oracle queries. This chapter is based on a paper to be published at an international conference but portions have been adapted from the author's Masters' thesis published on the arXiv:

- Hector B. Hougaard. "How to Generate Pseudorandom Permutations Over Other Groups: Even-Mansour and Feistel Revisited." In: *CoRR* abs/1707.01699 (2017). URL: http://arxiv.org/abs/1707.01699

Reproduced with permission from the author.

Please consult Figure 1.1 for an overview of our results.

1.6 OUTLINE

We start with a preliminary part that contains notions and definitions used in the various chapters including definitions of group actions, key exchanges, signature schemes, pseudorandomness, SIDH, R-LWE, EM, and our security model. Our results are spread throughout Pt. ii with GKEs from isogenies in Chp. 3, GKEs from R-LWE in Chp. 4, a GKE

compiler in Chp. 5, and the group action Even-Mansour in Chp. 6. Finally, in Pt. iii, we conclude by summarizing our results, discussing their relation to the wider field of cryptography, and considering future work based on them. Each part has been organized linearly and the entire dissertation is meant to build linearly as well.

2

PRELIMINARIES

In this chapter, we will present notation, definitions, and basic results. This chapter will serve as a basis for understanding the results in the dissertation and a reference for what is used later.

2.1 NOTATION

We assume knowledge of some standard set and probability theory for our notation. When χ is a probability distribution over a set S, we let $s \xleftarrow{R} \chi$ denote the process of sampling an element $s \in S$ according to the distribution χ. As a special case, we let $s \xleftarrow{R} S$ denote the process of choosing an element $s \in S$ uniformly at random from a set S. We denote an algorithm as Algo and denote the output of y from the algorithm when given the input x as $y \leftarrow \mathsf{Algo}(x)$. As a combination of notations, if the algorithm is probabilistic, we may draw attention to this using the notation $y \xleftarrow{R} \mathsf{Algo}(x)$. Also, as a general shorthand common to mathematics, wlog stands for "without loss of generality".

2.2 ALGEBRA

In order to explain the cryptographic background for our results, we will first need a mathematical foundation on which to build. In this dissertation, the mathematical foundation is algebra. We start with the concept of a group, one of the simplest constructions in mathematics.

Definition 1. *A **group** is defined over a set G with a binary operation $\star$ such that*

$$G \times G \to G,$$
$$(a, b) \mapsto a \star b,$$

and the binary operation satisfies the following axioms:

ASSOCIATIVITY $(a \star b) \star c = a \star (b \star c)$ *for all $a, b, c \in G$.*

IDENTITY *there exists an element e in G, called the **identity**, such that for all $a \in G$, $a \star e = a = e \star a$.*

INVERSE *for each $a \in G$ there exists an element $a^{-1} \in G$ such that $a^{-1} \star a = e = a \star a^{-1}$. Such an a^{-1} is called the **inverse** of a.*

*Furthermore, if it holds that $a \star b = b \star a$ for all $a, b \in G$, then the group is said to be **commutative**. We denote the number of elements in the group as $|G|$, called the **order** of the group.*

Example 2. *The set of integers,* $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ *can be made into a commutative group using the binary operation of addition,* $+$*, a so-called* ***additive group****. Here the identity element is* 0 *and the inverse of any element* $a \in \mathbb{Z}$ *is* $-a$*. The set of rational numbers* $\mathbb{Q} = \{a/b | b \neq 0\}$ *can be made into a group with the binary operation of multiplication,* $\cdot$*, a so-called* ***multiplicative group****. Here the identity element is* 1 *and the inverse of* $a \in G \setminus \{0\}$ *is* $1/a$*.*

We denote the successive operation of $g \in G$ on itself n times as $n \cdot g = g + g + \cdots + g$ for additive groups and $g^n = g \cdot g \cdot \cdots \cdot g$ for multiplicative groups, when n is a positive integer. The **order** of an element $g \in G$ is the smallest positive integer n such that $n \cdot g = e$, respectively $g^n = e$.

If a group can be generated by a single element g, i.e. for any $h \in G$ there exists a positive integer n such that $n \cdot g = h$ (or $g^n = h$, in multiplicative groups), then the group is said to be *cyclic*, usually denoted $G = \langle g \rangle$, where g is called the *generator* of G. In this case, if $|G| = n$, then the order of g is also n.

Example 3. *We can define an operation* $\oplus$ *on the set of bits,* $\{0, 1\}$*. It takes two bits and outputs* 0 *if both bits are identical, and outputs* 1 *if the bits are different. We call this binary operation exclusive-OR (XOR). The following table shows how XOR functions.*

$\oplus$	0	1
0	0	1
1	1	0

Table 2.1: Exclusive-OR (XOR) operation.

We may use this binary operation to turn the set of bit strings of length n*, i.e.* $\{0, 1\}^n = \{(b_{n-1}b_{n-2} \cdots b_1 b_0) | b_i \in \{0, 1\}\}$*, into a group by using XOR bitwise. This means that each bit in a string is XOR'ed with the corresponding bit in the other string. Using this definition, a string is its own inverse.*

A group is a quite fundamental structure because of its simple definition, but many of the deepest theorems in algebra are deduced simply from the implications of their structure. We shall not go too deep into group theory, but let us give an easy proposition giving some of the properties we will need later. As this is the preliminary chapter, we refrain from giving a proof but remark that each may easily be proven from the definition alone.

We sometimes write $(G, \star)$ for the group, in order to explicitly state that the group is made up of the set G and the binary operation $\star$. Usually, we use the binary operator $+$ to denote an additive group and the binary operator $\cdot$ to denote a multiplicative group.

Proposition 4. *If $(G, \star)$ is a group then*

1. *the identity e is unique,*
2. *for each $a \in G$, a^{-1} is uniquely defined,*
3. *$(a^{-1})^{-1} = a$ for all $a \in G$,*
4. *$(a \star b)^{-1} = (b^{-1}) \star (a^{-1})$,*
5. *for any $a_1, a_2, \ldots, a_n \in G$, the value of $a_1 \star a_2 \star \cdots \star a_n$ is independent of how parenthesis are set.*

The examples above showed that well-known sets can be made into groups. However, some groups can be given a second binary operation that turns the group into a ring.

Definition 5. *A **ring** is a commutative group $(R, +)$ with a secondary binary operation $\cdot$ such that the following axioms hold:*

ASSOCIATIVITY *$(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in R$.*

DISTRIBUTIVITY *For all $a, b, c \in R$,*

$$(a + b) \cdot c = a \cdot c + b \cdot c \quad \text{and} \quad a \cdot (b + c) = a \cdot b + a \cdot c.$$

*The ring is called **commutative** if multiplication is commutative and is said to have an **identity**, if there is an element $1 \in R$ for which*

$$1 \cdot a = a = a \cdot 1 \quad \text{for all } a \in R.$$

We will occasionally skip the $\cdot$ notation and simply write $a \cdot b$ as ab for $a, b \in R$. We may also denote the group or ring simply by G or R when the operations are understood.

Definition 6. *A commutative ring R with identity $1 \neq 0$ for which every element has a multiplicative inverse, i.e. for each $a \in R$ there exists an element $b \in R$ such that $ab = ba = 1$, is called a **field**.*

Example 7. *The set of integers $\mathbb{Z}$ defined in Example 2 is a ring under the group operation of addition and the ring operation of multiplication (having the identity 1). However, it is not a field as not every element has a multiplicative inverse. The set of rational numbers $\mathbb{Q}$ under the same operations, on the other hand, is not only a ring but a field too.*

Example 8. *Let us consider a subset of the integers, namely the set $\mathbb{Z}/n\mathbb{Z} = \{0, 1, 2, \ldots, n-1\}$. We can turn this set into a field, the **ring of integers modulo** n. Let $n = 0, n+1 = 1, n+2 = 2, \ldots$, then $\mathbb{Z}/n\mathbb{Z}$ is a ring under addition modulo n and multiplication modulo n. For example, 5 modulo 2 is equal to 1.*

Addition modulo n and multiplication modulo n work the same as regular addition and multiplication on the integers, however we have the following distributive properties:

$$a + b \pmod{n} = (a \pmod{n} + b \pmod{n}) \pmod{n}, \text{ and}$$
$$a \cdot b \pmod{n} = (a \pmod{n} \cdot b \pmod{n}) \pmod{n}.$$

If $a \pmod{n} = b \pmod{n}$, *we may also write* $a \equiv b \pmod{n}$. *When* $n = p$ *is a prime,* $\mathbb{Z}/p\mathbb{Z}$ *is a field having order* $|\mathbb{Z}/p\mathbb{Z}| = p - 1$.

Groups and rings in themselves are quite interesting but sometimes we want to look at the relationship between different groups or between different rings. For sets we have functions but for groups and rings we have homomorphisms.

Definition 9. *A map between groups (rings) that preserves the group (ring) operation, is said to be a group (ring)* ***homomorphism****. More specifically, for the groups* $(G, +)$ *and* $(G', \cdot)$, *it holds for a homomorphism* $\phi : G \to G'$ *that*

$$\phi(g + h) = \phi(g) \cdot \phi(h),$$

for all $g, h \in G$. *A bijective group (ring) homomorphism is called a group (ring)* ***isomorphism****.*

The elements $\{g \in G : \phi(g) = 1_{G'}\}$ *is called the* ***kernel*** *of* ϕ *and denoted* $\ker(\phi)$.

Lastly, we will need the definition of a group action. Group actions are generalizations of group operations in that a group operation acts on group elements, taking one group element and letting it "act" on another group element via the group operation. A group action however, is defined for a group acting on a set. If the set is equal to the group, then the group action is simply the group operation. Let us see this in more detail.

Definition 10. *A* ***group action*** *is defined for a group* $(G, \cdot)$ *on a set* X *such that*

$$G \times X \to X,$$
$$(g, x) \mapsto g \star x,$$

and the map satisfies the following properties:

COMPATIBILITY $g \star (h \star x) = (g \cdot h) \star x$ *for all* $g, h \in G, x \in X$.

IDENTITY $1 \star x = x$ *for all* $x \in X$.

Furthermore, a group action may satisfy the following properties:

TRANSITIVE *For every* $(x, y) \in X \times X$, *there is a* $g \in G$ *such that* $g \star x = y$.

FREE *For* $x \in X$ *and* $g, h \in G$ *satisfying* $g \star x = h \star x$ *we have* $g = h$.

*A group action is said to be **regular** if it is both transitive and free.*

If we let $\sigma_g = g \star x$ for all $x \in X$, then σ_g is a permutation of X. Any group operation on a group G can be trivially defined as a group action $G \times G \to G$. All such group actions are regular.

2.3 CRYPTOGRAPHY

Secrets have always required secrecy. Although cryptography has its foundations in trying to relay secrets between parties and hiding away information, it has evolved quite a bit since its simple cipher past. Nowadays, any computer can unscramble letters that have been rearranged, so we need to be more clever. This is the realm of modern cryptography, using computers to guard secrets against malicious parties who also have access to computers, so called adversaries.

For a cryptographic construction, we must consider how to give a guarantee of its security, a proof. At the advent of modern cryptography, this guarantee was perfect security, i.e. that even a single bit of a message could only be deciphered correctly with probability $1/2$ without the use of the secret key, a mere coin toss. Unfortunately, this forced the key to be as long as the message, a problem when randomness is expensive to create and messages can vary in length almost arbitrarily. The solution was to loosen the security notion slightly, from perfect indistinguishability to *computational* indistinguishability, i.e. we now consider the computational limits of an adversary.

Computers follow logic operations and the combination of logic operations constitute an algorithm. At the same time, mathematics is built from basic logic operations, and so, many mathematical problems can be defined for computers. Likewise, if there exists a mathematical solution to a logical or mathematical problem, we can potentially create a computer algorithm to solve the problem.

However, physical computers can only compute a limited number of operations per second, so we must consider the amount of time an algorithm needs in order to compute an answer. This is where our notion of a "hard" problem comes from. If every computer or algorithm - with some definition of physical capability - cannot solve the problem in at most polynomial-time, the problem is considered a hard problem.

Definition 11 (Efficient algorithm; [40], p. 47)**.** *An algorithm $\mathcal{A}$ runs in polynomial-time if there exists a polynomial p such that, for every input $x \in \{0,1\}^n$, the computation of $\mathcal{A}(x)$ terminates within at most $p(|x|)$ steps, where $|x|$ denotes the length of the string x.*

Polynomial-time is simply a measure of efficiency. As there are many other types of functions, we have an ordering of function-types. We usually compare function-types using what is called the **big O notation**. For the purpose of explaining orders of function-types, we

write that $f(n) = O(g(n))$ if the positive function $f(n)$ is a positive constant multiple of $g(n)$, i.e. $f(n) \leq M \cdot g(n)$, for some $M \in \mathbb{R}_+$ and for sufficiently large values of $n \in \mathbb{Z}$, i.e. all $n > N_0$ for some $N_0 \in \mathbb{Z}_+$. As algorithms can be analysed by the number of steps/processes they use, we can compare algorithms by the function-types that the number of steps correspond to, i.e. their computational complexity.

Example 12. *We have that for (positive) constant functions (e.g. $f(n) = 2$), logarithmic functions (e.g. $g(n) = \log(n)$), linear functions (e.g. $h(n) = 10 \cdot n$), polynomial functions (e.g. $j(n) = n^c$ for some $c \geq 1$), sub-exponential functions (e.g. $k(n) = n^{\log(n)}$), and exponential functions (e.g. $l(n) = e^n$),*

$$2 = O(\log(n)) = O(10 \cdot n) = O(n^c) = O(n^{\log(n)}) = O(e^n).$$

In some sense, the orders of complexity follow the pattern

$$\textit{constant} < \textit{logarithmic} < \textit{linear} < \textit{polynomial} < \textit{sub-exponential} < \textit{exponential}.$$

A "hard" problem is then a problem where the probability that a probabilistic polynomial-time (PPT) adversary can solve (or succeed at) it is negligible. PPT adversaries are algorithms that run in polynomial-time. They are usually assumed to have access to some source of randomness that might make their attacks more successful, hence "probabilistic".[1] Here, "negligibly" means:

Definition 13 ([40], Defn. 3.4, p. 48)**.** *A function f from the natural numbers to the non-negative real numbers is negligible if for every positive polynomial p there is an N such that for all integers $n > N$ it holds that $f(n) < 1/p(n)$.*

Hard problems themselves are not cryptographic protocols. Usually, they are mathematical or algorithmic problems that do not have any polynomial-time (or better) solutions. It is therefore a question of trust in the hardness of a problem, due to time and/or attacks that fail, that determines the worth of a hard problem. But anyone could simply construct a cryptographic protocol and claim hardness. This is where reduction proofs come in. A reduction proof, or simply a reduction, is a proof-theoretic construction that explicitly links a protocol to a hard problem. The longer the hard problem has been around and the tighter the reduction, the stronger the security of the protocol. It is therefore necessary to explain how such reductions work in practice.

Firstly, an algorithm that can distinguish between the protocol output and a random output in polynomial-time is assumed to exist. Secondly, an instance of the protocol is created such that distinguishing between the protocol output and a random output solves the hard problem. Per assumption the problem was hard, i.e. could not be

1 Adversaries are not forced to employ randomness. If they do not, we sometimes call them "deterministic".

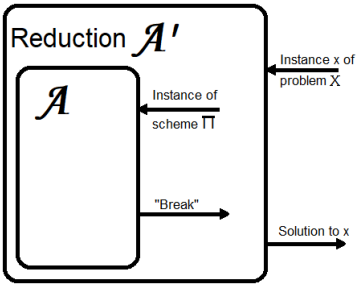

Figure 2.1: Reduction proof overview [40]

solved by any polynomial-time algorithm, hence a contradiction is obtained. This means that the protocol itself is secure. See Figure 2.1 for an overview of this construction.

More specifically, assume that a problem X cannot be solved by any polynomial-time algorithms. As an example, consider a polynomial-time prime factorization of large numbers, which is considered hard for classical computers. If we want to show that a cryptographic protocol Π is secure (according to some precise definition), assume for contradiction that there exists some (probabilistic) polynomial-time algorithm $\mathcal{A}$ that "breaks" Π with non-negligible success probability $\varepsilon(n)$, where n is the security parameter (a variable adjusting the protocol instantiation).

We then construct an efficient algorithm $\mathcal{A}'$, a "reduction", which will try to solve the problem X using $\mathcal{A}$ as a sub-algorithm. It must be noted that $\mathcal{A}'$ only uses $\mathcal{A}$ as a black box, i.e. it has no idea how $\mathcal{A}$ works internally, only what it requires for inputs and what it returns for outputs. For an instance x of X, two properties must then hold:

1. As far as $\mathcal{A}$ can tell, it is interacting with a valid instance of Π, i.e. all inputs to $\mathcal{A}$ are identically distributed as in an instance of Π.

2. If $\mathcal{A}$ does succeed in breaking the given instance of Π simulated by $\mathcal{A}'$, this should then allow $\mathcal{A}'$ to solve the instance x with probability greater than $1/p(n)$, for some polynomial function $p(\cdot)$.

These two properties then imply that $\mathcal{A}'$ solves instance x of X with probability $\varepsilon(n)/p(n)$, which is non-negligible. This means that an instance of the hard problem X has been solved in polynomial-time with non-negligible probability, a contradiction. Hence, Π must be secure.

Alas, we must warn that using a reduction proof as detailed above will only ever give a cryptographic scheme that is secure as long as the hard problem is hard. This is exactly the problem for many of the constructions that are now vulnerable due to the advent of quantum computing. One of these vulnerable constructions is the

following Diffie-Hellman key exchange that relies on the hardness of the decisional Diffie-Hellman problem.

2.3.1 *Diffie-Hellman*

Consider the group $\mathbb{Z}_q$, the group of integers modulo q (see Example 8), i.e. $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z} = \{0, 1, \ldots, q-1\}$. The Diffie-Hellman key exchange protocol is a one round, two-party key exchange protocol that consists of a security parameter 1^λ (we use the cryptographic convention of inputting the value λ using a string of 1s of length λ) as well as a parameter generating algorithm, Gen, that given the security parameter 1^λ as input, outputs a description of a cyclic group G along with its order q and a generator $g \in G$.

Protocol 14 (Diffie-Hellman key exchange (DH))**.** *For parties $\mathcal{P}_i$, $i = 0, 1$, we define the DH protocol is as follows:*

setup: *For security parameter 1^λ, Gen outputs to both parties the public parameter tuple: $\mathfrak{P} = (G, q, g) \leftarrow \mathsf{Gen}(1^\lambda)$.*

publish: *Each party $\mathcal{P}_i$ begins by choosing a uniform secret key, $x_i \xleftarrow{R} \mathbb{Z}_q$, and computes their public key, $h_i = g^{x_i}$, before sending it to the other party.*

keygen: *Upon receiving key h_{1-i}, party $\mathcal{P}_i$ uses their secret key x_i to compute the shared key $k_i := h_{1-i}^{x_i}$.*

The protocol satisfies correctness, i.e. $k_0 = k_1 = k$.

We state without proof the well-known fact that the passive adversary (eavesdropper) security of DH reduces to the following classical hard problem.

Definition 15 (Decisional Diffie-Hellman (DDH) problem)**.** *Given a uniformly sampled tuple from one of the following two distributions:*

- *$(\mathfrak{P}, (g^{x_0}, g^{x_1}), k)$, where $\mathfrak{P} = (G, q, g)$ for a cyclic group G of order q and generated by g, uniformly sampled $x_0, x_1 \xleftarrow{R} \mathbb{Z}_q$, and $k = g^{x_0 x_1} \in G$ is the shared key of the DH protocol (Protocol 14),*
- *$(\mathfrak{P}, (g^{x_0}, g^{x_1}), k')$, where $\mathfrak{P} = (G, q, g)$ for a cyclic group G of order q and generated by g, uniformly sampled $x_0, x_1 \xleftarrow{R} \mathbb{Z}_q$, and $z \xleftarrow{R} \mathbb{Z}_q$ sampled uniformly such that $k' = g^z$ is uniformly distributed in G,*

determine which the tuple is sampled from.

2.3.2 *Signature schemes*

We also provide a definition of a (digital) signature scheme as we will assume the existence of such cryptographic constructions for our authenticated group key exchanges.

Definition 16 ([40], Defn. 12.1, p. 442)**.** *A **(digital) signature scheme**, Π_{sign}, consists of the PPT algorithms* Gen, Sign, *and* Vrfy *used in the following way:*

setup: *The key-generation algorithm* Gen *takes security parameter 1^λ as input and outputs the pair of keys* (sign, vrfy)*, the **signing key** (or private key) and **verification key** (or public key), respectively. We assume that* sign *and* vrfy *each has length at least λ, and that λ can be determined from* sign *or* vrfy*.*

sign: *The signing algorithm* Sign *takes as input a signing key* sign *and a message m from some message space (that may depend on* vrfy*). It outputs a signature σ, and we write this as $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sign}}(m)$.*

verify: *The deterministic verification algorithm* Vrfy *takes as input a verification key* vrfy*, a message m, and a signature σ. It outputs a bit b, with $b = 1$ meaning **valid** and $b = 0$ meaning **invalid**. We write this as $b \leftarrow \mathsf{Vrfy}_{\mathsf{vrfy}}(m, \sigma)$.*

For correctness, it is required that except with negligible probability over (sign, vrfy) *output by* $\mathsf{Gen}(1^\lambda)$*, it holds that* $1 \leftarrow \mathsf{Vrfy}_{\mathsf{vrfy}}(m, \mathsf{Sign}_{\mathsf{sign}}(m))$ *for every (legal) message m.*

Definition 17 (Security of signatures; [40], Defn. 12.2, p. 443)**.** *The signature experiment* $\mathsf{Sig}\text{-}\mathsf{forge}_{\mathcal{A},\Pi_{sign}}(1^\lambda)$*:*

1. $\mathsf{Gen}(1^\lambda)$ *is run to obtain keys* (sign, vrfy)*.*
2. *Adversary $\mathcal{A}$ is given* vrfy *and access to an oracle* $\mathsf{Sign}_{\mathsf{sign}}(\cdot)$*. The adversary then outputs (m, σ). Let $\mathcal{Q}$ denote the set of queries that $\mathcal{A}$ asked its oracle.*
3. *$\mathcal{A}$ **succeeds** if and only if* (1) $1 \leftarrow \mathsf{Vrfy}_{\mathsf{vrfy}}(m, \sigma)$ *and* (2) $m \notin \mathcal{Q}$*. In this case, the output of the experiment is defined to be* 1*.*

A signature scheme $\Pi_{sign}(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is **existentially unforgeable under an adaptive chosen-message attack**, or just **secure**, if for all PPT adversaries $\mathcal{A}$, there is a negligible function negl such that:*

$$Succ_{\mathcal{A}}(\Pi_{sign}) = Pr[\mathsf{Sig}\text{-}\mathsf{forge}_{\mathcal{A},\Pi_{sign}}(1^\lambda) = 1] \leq negl(\lambda).$$

2.3.3 *Pseudorandomness*

For our work with the Even-Mansour construction in Chp. 6, we will need the definition of a pseudorandom permutation, which we give below.

A pseudorandom permutation is a permutation selected from a family of permutations by a key. We therefore talk about a keyed permutation, $P : K \times X \to X$. In order to prove the pseudorandomness

of the keyed permutation, we intend to play a game of determining "pseudorandom or random permutation", i.e. distinguishability, but for permutations. $\mathcal{A}$ is given a permutation oracle $\overline{P}$ which is chosen randomly (with equal probability) from the following two distributions:

1. A random key $k \xleftarrow{R} K$ is uniformly chosen and used to permute via $\overline{P}(x) := P(k, x)$ for all $x \in X$, or
2. A random permutation $\pi \xleftarrow{R} \mathsf{Perm}_{X \to X}$ is chosen and used to permute via $\overline{P}(x) := \pi(x)$ for all $x \in X$.

The adversary "wins" the game if it can distinguish from which distribution $\overline{P}$ was chosen, with probability significantly better than a fair coin toss. We denote the case where an adversary $\mathcal{A}$, having access to oracles (possibly including an auxiliary collection of oracles, $\mathfrak{H}(\cdot)$), wins the game with security parameter 1^λ, as $\mathcal{A}^{\mathfrak{H}(\cdot)}_{\mathfrak{G}(\cdot)}(1^\lambda) = 1$, where $\mathfrak{G}(\cdot)$ is the collection of game-specific oracles, e.g. $P(k, \cdot)$ for a fixed k or $\pi(\cdot)$.

We now redefine the classical bit string notion of pseudorandomness (and superpseudorandomness) to arbitrary sets.[2]

Definition 18. *Let $P : K \times X \to X$ be an efficient permutation, keyed by a group K. P is called a **pseudorandom permutation (PRP)** if for all adversaries $\mathcal{A}$, limited to only polynomially-many queries to a permutation oracle and possibly to the collection of auxiliary oracles $\mathfrak{H}(\cdot)$, we have that*

$$\left| \Pr_{k \xleftarrow{R} K} \left[\mathcal{A}^{\mathfrak{H}(\cdot)}_{P(k,\cdot)}(1^\lambda) = 1 \right] - \Pr_{\pi \xleftarrow{R} \mathsf{Perm}_{X \to X}} \left[\mathcal{A}^{\mathfrak{H}(\cdot)}_{\pi(\cdot)}(1^\lambda) = 1 \right] \right|$$

is negligible, where 1^λ is the security parameter and $\mathsf{Perm}_{X \to X}$ is the set of permutations on X.

Giving the adversary access to an inverse permutation oracle gives us a stronger notion of pseudorandomness.

Definition 19. *Let $P : K \times X \to X$ be an efficient permutation, keyed by a group K. P is said to be a **super pseudorandom permutation (SPRP)** if for all adversaries $\mathcal{A}$, limited to only polynomially-many queries to the permutation and inverse permutation oracles and possibly to a collection of auxiliary oracles, $\mathfrak{H}(\cdot)$, we have that*

$$\left| \Pr_{k \xleftarrow{R} K} \left[\mathcal{A}^{\mathfrak{H}(\cdot)}_{P(k,\cdot), P^{-1}(k,\cdot)}(1^\lambda) = 1 \right] - \Pr_{\pi \xleftarrow{R} \mathsf{Perm}_{X \to X}} \left[\mathcal{A}^{\mathfrak{H}(\cdot)}_{\pi(\cdot), \pi^{-1}(\cdot)}(1^\lambda) = 1 \right] \right|$$

is negligible, where 1^λ is the security parameter and $\mathsf{Perm}_{X \to X}$ is the set of permutations on X.

We call a (super) pseudorandom permutation $P : K \times X \to X$ a (super) pseudorandom permutation on X.

2 As given in e.g. Definition 3.28, p. 80 in [40]

2.4 BURMESTER-DESMEDT GROUP KEY EXCHANGES

We now define the Burmester-Desmedt I (BDI) protocol [8], which is a two-round group key exchange (GKE) based on DH. Here, a *round* necessitates the exchange of new information, such as a public key or specific message, in order for a party to proceed. We give the same description as given by Hatano, Miyaji, and Sato [30].

Protocol 20 (Burmester-Desmedt I (BDI))**.** *For parties $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{n-1}$, arranged in a ring[3] (see Figure 2.2), we define the BDI protocol as follows:*

setup: *Given the security parameter 1^λ, outputs a description of a group G of order prime p and generator g.*

publish$_1$: *Each $\mathcal{P}_i$ computes a public key $z_i = g^{r_i}$ corresponding to a uniformly random, secretly chosen $r_i \xleftarrow{R} \mathbb{Z}_p^*$, and sends it to $\mathcal{P}_{i-1}$ and $\mathcal{P}_{i+1}$.*

publish$_2$: *Each $\mathcal{P}_i$ then computes $x_{[i-1,i+1]} = \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i} = g^{r_i r_{i+1} - r_i r_{i-1}}$ and broadcasts to this value all other parties.*

keygen: *Each $\mathcal{P}_i$ finally computes the shared key,*

$$\begin{aligned} K &= (z_{i-1})^{n r_i} \cdot x_{[i-1,i+1]}^{n-1} \cdot x_{[i,i+2]}^{n-2} \cdots x_{[i-3,i-1]} \\ &= g^{r_1 r_2 + r_2 r_3 + \cdots + r_n r_1}. \end{aligned}$$

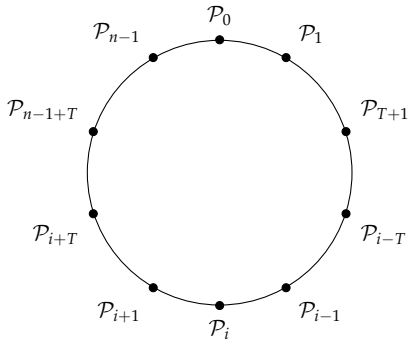

Figure 2.2: BDI circle structure

We can think of BDI as a GKE having a closed-chain-like structure, each party being a link connected to both the previous and next party.

Another two-round Burmester-Desmedt GKE, the Burmester-Desmedt II (BDII) protocol [9], instead utilizes a tree structure. For pedagogical reasons, we describe the binary double-tree version. Here, the parties are assumed to be arranged in two trees, connected at the roots, as shown in Figure 2.3. Each party node branches into two and we call the parties on the lowest level of the tree the *leaves*.

3 The indexes are taken modulo n so that $\mathcal{P}_0 = \mathcal{P}_n$ and $\mathcal{P}_{n+1} = \mathcal{P}_1$

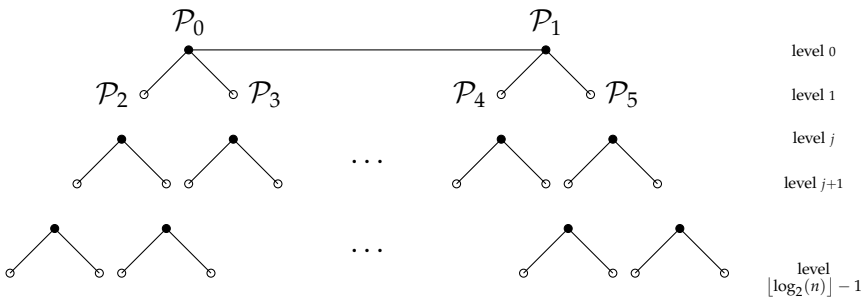

Figure 2.3: BDII tree-based structure

Each party $\mathcal{P}_i$ has a parent $\mathsf{par}(i)$, left child $L.\mathsf{cld}(i)$, and right child $R.\mathsf{cld}(i)$, the collection of which we call the *neighbours* of $\mathcal{P}_i$. For leaves, the only neighbour is the parent. We define $\mathsf{ancestors}(i)$ to be the set of the indexes of all ancestors (parents, grandparents, great grandparents, etc.) of a party $\mathcal{P}_i$, including i but removing 0 and 1. $\mathcal{P}_0$ and $\mathcal{P}_1$ are defined to be each other's parent.

The following definition for BDII is also as given by Hatano, Miyaji, and Sato [30]. For the protocol, we assume that n parties $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{n-1}$ wish to generate a shared key.

Protocol 21 (Burmester-Desmedt II (BDII))**.** *Assume the parties $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{n-1}$ are arranged in a binary double-tree.*

setup: *For security parameter 1^λ, outputs a group G of order prime p and generated by g.*

publish$_1$: *Each party $\mathcal{P}_i$ computes a public key $z_i = g^{r_i}$ corresponding to a uniformly random, secretly chosen $r_i \xleftarrow{R} \mathbb{Z}_p^*$, and sends it to its neighbours.*

publish$_2$: *Each party $\mathcal{P}_i$ then computes both*

$$x_{L.\mathsf{cld}(i)} = \left(\frac{z_{\mathsf{par}(i)}}{z_{L.\mathsf{cld}(i)}}\right)^{r_i} = g^{r_i r_{\mathsf{par}(i)} - r_i r_{L.\mathsf{cld}(i)}} \text{ and}$$

$$x_{R.\mathsf{cld}(i)} = \left(\frac{z_{\mathsf{par}(i)}}{z_{R.\mathsf{cld}(i)}}\right)^{r_i} = g^{r_i r_{\mathsf{par}(i)} - r_i r_{R.\mathsf{cld}(i)}}$$

and multicasts[4] the respective values to all of its left, respectively right, descendants.[5]

keygen: *Each $\mathcal{P}_i$ may finally compute the shared key*

$$K = (z_{\mathsf{par}(i)})^{r_i} \cdot \prod_{j \in \mathsf{ancestors}(i)} x_j = g^{r_0 r_1}.$$

Theorem 22 (Security of BDII [18])**.** *Assuming the DDH problem over the group G is hard, BDII is a secure GKE protocol.*

4 Sends only to designated parties.

5 This means that a party $\mathcal{P}_i$ sends $x_{L.\mathsf{cld}(i)}$ to its left child, all the left child's children, all their children, and so forth. Respectively, it sends $x_{R.\mathsf{cld}(i)}$ to its right child, all the right child's children, all their children, and so forth.

2.5 SUPERSINGULAR ISOGENY DIFFIE-HELLMAN

Although it is rather complicated mathematics, we must (briefly) cover the basics of elliptic curves we will need in this dissertation.

Let E be an elliptic curve defined over a finite field of order q, $\mathbb{F}_q$, with $q = p^2$ for a prime p, e.g. elliptic curves in Weierstrass form $E : y^2 = x^3 + Ax + B$, where $A, B \in \mathbb{F}_q$. As this is a function, we may consider the solutions over some field, in this case $\mathbb{F}_q$, which generate the elliptic curve group, $E(\mathbb{F}_q)$, with identity element $\mathcal{O}_E$. We call E *supersingular* if the p torsion group, $E[p] = \{X \in E(\overline{\mathbb{F}}_q) \mid pX = \mathcal{O}_E\}$, is equal to $\{\mathcal{O}_E\}$. For Weierstrass forms, the j-invariant may be given by the equation

$$j(E) = 1728\frac{4A^3}{4A^3 - 27B^3} \in \mathbb{F}_q.$$

We call two elliptic curves E_0 and E_1 *isomorphic*, written $E_0 \cong E_1$, if and only if $j(E_0) = j(E_1)$. We note that since the j-invariants lie in $\mathbb{F}_q$, both addition and multiplication of j-invariants is taken over $\mathbb{F}_q$.

An (separable) isogeny $\phi : E_0 \to E_1$ of elliptic curves over the field $\mathbb{F}_q$, is a rational map[6] inducing a homomorphism between the elliptic curve groups $E_0(\overline{\mathbb{F}_q})$ and $E_1(\overline{\mathbb{F}_q})$. A separable isogeny is either surjective or the 0-map. It has a finite kernel and sends $\mathcal{O}_{E_0(\mathbb{F}_q)}$ to $\mathcal{O}_{E_1(\mathbb{F}_q)}$. If such an (non-zero) isogeny exists, the elliptic curves E_0, E_1 are said to be isogenous. The kernel of an isogeny ϕ is

$$\ker(\phi) = \{P \in E_0(\mathbb{F}_q) \mid \phi(P) = \mathcal{O}_{E_1}\}.$$

This set creates a finite (sub)group and, using Vélu's formulas [63, pp. 392-393], may be used to explicitly compute an isogeny. On the other hand, for a given isogeny $\phi : E_0 \to E_1$, there exists a corresponding finite subgroup $\langle\Phi\rangle$ such that $\ker(\phi) = \langle\Phi\rangle$. When the isogeny is separable the degree of the isogeny will be equal to the order of the kernel, i.e. $\deg(\phi) = \#\langle\Phi\rangle = \ell$. We may then call the isogeny an ℓ-isogeny and that E_0 is ℓ-isogenous to E_1.

Given two isogenous elliptic curves, finding an isogeny between them is called the isogeny finding problem. This problem is hard even for quantum computers, as the fastest known algorithm takes exponential time when the curves involved are supersingular. Therefore, cryptosystems based on the isogeny finding problem can provide quantum resistance. One such quantum-secure cryptosystem is the supersingular isogeny Diffie-Hellman (SIDH) key exchange by Jao and De Feo [25, 38].[7]

6 This is a geometric algebraic notion that requires more background than we are willing to give in this dissertation. Although rational maps are not as simple as rational functions, it will suffice to think of them as consisting of such for this dissertation.

7 For a description of the protocol, we use subgroups generated by $P + [r]Q$ instead of $[n]P + [m]Q$, as per the supersingular isogeny key encapsulation mechanism (SIKE) protocol. See `https://sike.org/` for further details.

Consider parties $\mathcal{P}_0$ and $\mathcal{P}_1$. Given security parameter 1^λ, the public parameter generating algorithm GenSS outputs the tuple $(p, E, \{P_0, Q_0\}, \{P_1, Q_1\})$, where $p = f\ell_0^{e_0}\ell_1^{e_1} \pm 1$ is prime for a small integer $f > 0$ and $\ell_0^{e_0} \approx \ell_1^{e_1}$ (usually $\ell_0 = 2$ and $\ell_1 = 3$), E is a randomly chosen supersingular elliptic curve over $\mathbb{F}_q = \mathbb{F}_{p^2}$ such that $\#E(\mathbb{F}_q) = (p \pm 1)^2$, and $\{P_i, Q_i\}$ is a randomly chosen basis of $E[\ell_i^{e_i}]$ for $i \in \{0, 1\}$.

Protocol 23 (Supersingular isogeny Diffie-Hellman (SIDH); [25])**.** *The protocol is as follows.*

setup: *For security parameter* 1^λ, GenSS *outputs to both parties the tuple:*

$$\mathsf{params} := (p, E, \{P_0, Q_0\}, \{P_1, Q_1\}) \xleftarrow{R} \mathsf{GenSS}(1^\lambda).$$

publish: *Given* params *as input, party* $\mathcal{P}_0$ *randomly chooses* $r_0 \xleftarrow{R} \mathbb{Z}/\ell_0^{e_0}\mathbb{Z}$ *and computes* $R_0 := P_0 + [r_0]Q_0$*. It then computes the isogeny* $\phi_0 : E \to E_0 \cong E/\langle R_0 \rangle$ *having* $\ker(\phi_0) = \langle R_0 \rangle$*, along with the points* $\phi_0(P_1)$ *and* $\phi_0(Q_1)$*.* $\mathcal{P}_0$ *has secret and public keys*

$$sk_0 := r_0 \text{ and } pk_0 := (E_0, \phi_0(P_1), \phi_0(Q_1)),$$

respectively, where it sends pk_0 *to* $\mathcal{P}_1$*. Likewise,* $\mathcal{P}_1$ *computes the secret and public keys*

$$sk_1 := r_1 \text{ and } pk_1 := (E_1, \phi_1(P_0), \phi_1(Q_0)),$$

respectively, of which it sends pk_1 *to* $\mathcal{P}_0$*.*

keygen: *Party* $\mathcal{P}_0$ *takes* pk_1 *as input and computes an isogeny* $\phi_0' := E_1 \to E_{1,0}$ *with* $\ker(\phi_0') = \langle \phi_1(P_0) + [r_0]\phi_1(Q_0) \rangle = \langle \phi_1(R_0) \rangle$ *and computes* $K_0 = j(E_{1,0}) \in \mathbb{F}_q$*.*

Party $\mathcal{P}_1$ *likewise computes an isogeny* $\phi_1' := E_0 \to E_{0,1}$ *with* $\ker(\phi_1') = \langle \phi_0(P_1) + [r_1]\phi_0(Q_1) \rangle = \langle \phi_0(R_1) \rangle$ *and computes* $K_1 = j(E_{0,1}) \in \mathbb{F}_q$ *(see Figure 2.4).*

It holds that

$$E_{1,0} = \phi_0'(\phi_1(E)) \cong \phi_1'(\phi_0(E)) = E_{0,1},$$

i.e. $K_0 = j(E_{1,0}) = j(E_{0,1}) = K_1$*, such that* $\mathcal{P}_0$ *and* $\mathcal{P}_1$ *have the shared key* $K = K_0 = K_1$*.*

Except for notational changes, we give the following definitions ad verbum from De Feo, Jao, and Plût [25] .

Definition 24 (Decisional supersingular isogeny (DSSI) problem)**.** *Let* $p, \ell_0^{e_0}$*, and* E*, be as in Protocol 23. Let* E_0 *be another supersingular curve defined over* $\mathbb{F}_{p^2}$*. Decide whether* E_0 *is* $\ell_0^{e_0}$*-isogenous to* E*.*

8 Original TikZ code courtesy of De Feo via GitHub [17].

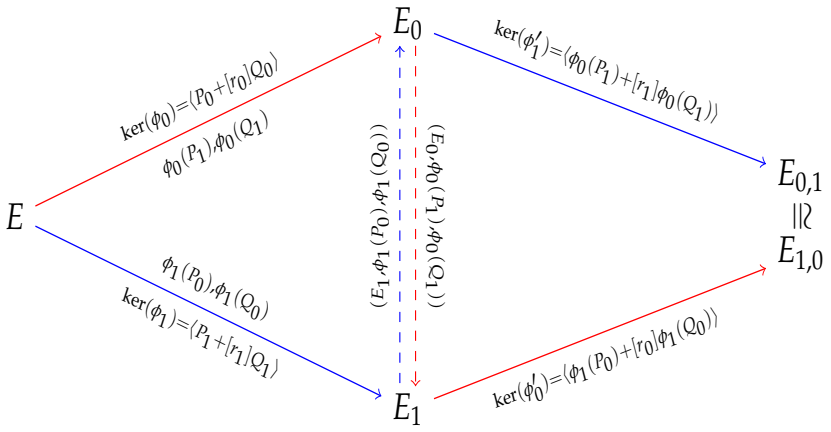

Figure 2.4: SIDH key exchange. Quantities only known by $\mathcal{P}_0$, respectively $\mathcal{P}_1$, are drawn in red, respectively blue. The dotted lines signify public keys being exchanged[8]

Definition 25 (Computational supersingular isogeny (CSSI) problem)**.** *Let $p, \ell_i^{e_i}, E$, and $\{P_i, Q_i\}$, for $i \in \{0,1\}$, be as in Protocol 23. Let $\phi_0 : E \to E_0$ be an isogeny whose kernel is $\langle P_0 + [r_0]Q_0\rangle$, where r_0 is chosen at random from $\mathbb{Z}/\ell_0^{e_0}\mathbb{Z}$. Given E_0 and the values $\phi_0(P_1), \phi_0(Q_1)$, find a generator R_0 of $\langle P_0 + [r_0]Q_0\rangle$.*

Definition 26 (Supersingular computational Diffie-Hellman (SSCDH) problem)**.** *Let $p, \ell_i^{e_i}, E$, and the basis $\{P_i, Q_i\}$, for $i \in \{0,1\}$, be as in Protocol 23. Let $\phi_0 : E \to E_0$ be an isogeny whose kernel is equal to $\langle P_0 + [r_0]Q_0\rangle$, and let $\phi_1 : E \to E_1$ be an isogeny whose kernel is $\langle P_1 + [r_1]Q_1\rangle$, where r_0 (respectively r_1) is chosen at random from $\mathbb{Z}/\ell_0^{e_0}\mathbb{Z}$ (respectively $\mathbb{Z}/\ell_1^{e_1}\mathbb{Z}$). Given the curves E_0, E_1 and the points $\phi_0(P_1), \phi_0(Q_1), \phi_1(P_0), \phi_1(Q_0)$, find the j-invariant of $E/\langle P_0 + [r_0]Q_0, P_1 + [r_1]Q_1\rangle$.*

Definition 27 (Supersingular decisional Diffie-Hellman (SSDDH) problem)**.** *Given a tuple sampled uniformly from one of the following two distributions:*

- $(E_0, E_1, \phi_0(P_1), \phi_0(Q_1), \phi_1(P_0), \phi_1(Q_0), E_{0,1})$, *where* $E_0, E_1, \phi_0(P_1)$, $\phi_0(Q_1), \phi_1(P_0)$, *and* $\phi_1(Q_0)$ *are as in the SSCDH problem and*

$$E_{0,1} \cong E/\langle P_0 + [r_0]Q_0, P_1 + [r_1]Q_1\rangle,$$

- $(E_0, E_1, \phi_0(P_1), \phi_0(Q_1), \phi_1(P_0), \phi_1(Q_0), E_x)$, *where* $E_0, E_1, \phi_0(P_1)$, $\phi_0(Q_1), \phi_1(P_0)$, *and* $\phi_1(Q_0)$ *are as in the SSCDH problem and*

$$E_x \cong E/\langle P_0 + [r_0']Q_0, P_1 + [r_1']Q_1\rangle,$$

  where r_0' (respectively r_1') is chosen at random from $\mathbb{Z}/\ell_0^{e_0}\mathbb{Z}$ (respectively $\mathbb{Z}/\ell_1^{e_1}\mathbb{Z}$),

determine which distribution the tuple is sampled from.

Theorem 28 (Security of SIDH; [25]). *Assuming the SSDDH problem is (quantum) hard, SIDH is a (post-quantum) secure key exchange (KE).*[9]

Note 29. *If $\mathcal{P}_0$ and $\mathcal{P}_1$ share a basis over $E[\ell^e]$, i.e. $P_0 = P_1 = P$ and $Q_0 = Q_1 = Q$, then $\mathcal{P}_0$ has public key $pk_0 := (E_0, \phi_0(P), \phi_0(Q))$ and $\mathcal{P}_1$ has public key $pk_1 := (E_1, \phi_1(P), \phi_1(Q))$. Any adversary wanting to find the secret key for one of the parties, say $\mathcal{P}_0$, needs to find $[x]$ from $\phi_i(P + [x]Q) = \phi_i(P) + [x]\phi_i(Q) = \mathcal{O}_{E_i}$, which is an instance of the so-called elliptic curve discrete logarithm problem. Although this problem is considered hard for classical adversaries, a quantum adversary can solve it in polynomial-time. Be aware that because of the adaptive attack of Galbraith, Petit, Shani, and Ti [28], it is important to choose random keys anew for each SIDH key exchange instance (see footnote 5 on page 82)*

For quantum adversaries, the SSCDH and SSDDH problems are equivalent, as shown by Galbraith and Vercauteren [29] and Thormarker [62].

2.6 RING-LEARNING-WITH-ERRORS (R-LWE)

Denote $[N] = \{0, 1, \ldots, N-1\}$. For the remainder of this dissertation, set $R = \mathbb{Z}[X]/(\Phi(X))$ for $\Phi(X) = X^m + 1$ for $m = 2^l$, for some $l \in \mathbb{Z}_+$. Let q be a positive integer and define the quotient ring $R_q = R/qR \cong \mathbb{Z}_q[X]/(\Phi[X])$, where $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$.

Traditionally, ring-learning-with-errors (R-LWE) based cryptography relies on the quantum hardness of the R-LWE problem, either the computational or distinguishable version. However, Bos, Costello, Naehrig, Stebila [7] were able to give a Diffie-Hellman-like definition of indistinguishability taking key reconciliation into account and also showed how the ring-learning-with-errors (R-LWE) based KE of Ding, Xie, and Lin [20] (with Peikert's reconciliation tweak [52]) reduces to their newly formulated problem. Newly defined problems do not inspire trust as they are seen as ad hoc solutions but Bos et al show that the Diffie-Hellman-like problem actually reduces to the decisional ring-learning-with-errors (D-R-LWE) problem, so any protocol reducing to their problem naturally also reduces to the D-R-LWE problem, giving quantum-security.[10] In order to define this new problem, the decision Diffie-Hellman-like (DDH-like) problem, we must therefore first define the D-R-LWE problem.

Definition 30 (Decisional ring-learning-with-errors (D-R-LWE) problem; [7], Definition 1). *Let m, R, q and R_q be as above. Let χ be a distribution over R_q and let $s \leftarrow \chi$. Define $O_{\chi,s}$ as an oracle that does the following:*

9 The result in [25] holds in the authenticated-links adversarial model of Canetti and Krawczyk [10].

10 For the search problem statement and the search-to-decision reduction (equivalence actually), see the original paper by Lyubashevsky, Peikert, and Regev [48].

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

The D-R-LWE problem for m, q, χ *is to distinguish* $O_{\chi,s}$ *from an oracle that returns uniformly random samples from* $R_q \times R_q$.

Note 31. *The D-R-LWE problem given above is in its normal form, that is to say that* s *is chosen randomly from the error distribution, not uniformly at random from* R_q. *However, this problem is as hard as choosing* s *uniformly at random from* R_q *[47, Lemma 2.24].*

To describe our R-LWE based GKE, we must introduce the R-LWE key exchange, on which it is based. For that purpose, we must first define Peikert's key reconciliation mechanism. Let $\lceil \cdot \rfloor$ denote the rounding function: $\lceil x \rfloor = z$ for $z \in \mathbb{Z}$ and $x \in [z - 1/2, z + 1/2)$.

Definition 32 ([7], Defn. 2)**.** *Let* q *be a positive integer. Define the* ***modular rounding*** *function*

$$\begin{aligned} \lceil \cdot \rfloor_{q,2} : \mathbb{Z}_q &\to \mathbb{Z}_2, \\ x &\mapsto \lceil x \rfloor_{q,2} = \left\lceil \tfrac{2}{q} x \right\rfloor \mod 2, \end{aligned}$$

and the ***cross-rounding*** *function*

$$\begin{aligned} \langle \cdot \rangle_{q,2} : \mathbb{Z}_q &\to \mathbb{Z}_2, \\ x &\mapsto \langle x \rangle_{q,2} = \left\lfloor \tfrac{4}{q} x \right\rfloor \mod 2. \end{aligned}$$

Both functions are extended to elements of R_q *coefficient-wise: for* $f = f_{m-1} X^{m-1} + \cdots + f_1 X + f_0 \in R_q$, *define*

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

We also define the ***randomized doubling*** *function*

$$\begin{aligned} \mathsf{dbl} : \mathbb{Z}_q &\to \mathbb{Z}_{2q}, \\ x &\mapsto \mathsf{dbl}(x) = 2x - e, \end{aligned}$$

where e *is sampled from* $\{-1, 0, 1\}$ *with probabilities* $p_{-1} = p_1 = \frac{1}{4}$ *and* $p_0 = \frac{1}{2}$.

As done with the rounding functions, we may apply the doubling function to elements in R_q by doing so coefficient-wise, resulting in polynomials in R_{2q}. We consider such an application to allow for odd q in the KE protocol. Applying instead both the rounding and doubling function on a uniformly random element in $\mathbb{Z}_q$ results in a uniformly random element in $\mathbb{Z}_{2q}$:

Lemma 33 ([7], Lem. 1)**.** *For odd q, if $v \in \mathbb{Z}_q$ is uniformly random and $\overline{v} \xleftarrow{R} \mathsf{dbl}(v) \in \mathbb{Z}_{2q}$, then, given $\langle \overline{v} \rangle_{2q,2}$, $\lceil \overline{v} \rfloor_{2q,2}$ is uniformly random.*

We can now define Peikert's reconciliation function, $\mathsf{rec}(\cdot)$. Given an element $w \in \mathbb{Z}_q$ that is "close" to a $v \in \mathbb{Z}_q$ and the cross-rounding of this v, the reconciliation function recovers $\lceil v \rfloor_{q,2}$.

Definition 34 (Peikert's reconciliation function)**.** *Define $I_0 = \{0, 1, \ldots, \lceil \frac{q}{2} \rfloor - 1\}$ and $I_1 = \{-\lceil \frac{q}{2} \rfloor, \ldots, -1\}$. Let $E = [-\frac{q}{4}, \frac{q}{4})$, then*

$$\begin{aligned} \mathsf{rec}: \quad & \mathbb{Z}_{2q} \times \mathbb{Z}_2 \to \mathbb{Z}_2, \\ & (w, b) \quad \mapsto \begin{cases} 0, \text{ if } w \in I_b + E \mod 2q, \\ 1, \text{ otherwise .} \end{cases} \end{aligned}$$

We can also consider the reconciliation of a polynomial in R_q by doing coefficient-wise reconciliation. The following lemma thereby allows us to reconcile two polynomials in R_q that are "close" to each other.

Lemma 35 ([7], Lem. 2)**.** *For odd q, let $v = w + e \in \mathbb{Z}_q$ for $w, e \in \mathbb{Z}_q$ such that $2e \pm 1 \in E \pmod q$. Let $\overline{v} = \mathsf{dbl}(v)$, then $\mathsf{rec}(2w, \langle \overline{v} \rangle_{2q,2}) = \lceil \overline{v} \rfloor_{2q,2}$.*

Finally, we come to the definition of the R-LWE key exchange. Let m, R, q, R_q and χ be as in the D-R-LWE problem (Definition 30). Given R_q, the parameter generating algorithm ParaGen outputs a uniformly random $a \xleftarrow{R} R_q$.

Protocol 36 (R-LWE key exchange (R-LWE KE))**.** *Parties $\mathcal{P}_0$ and $\mathcal{P}_1$ generate an instance of the R-LWE KE protocol Π as follows:*

setup: *For the input R_q, ParaGen outputs the public parameter $a \xleftarrow{R} \mathsf{ParaGen}(R_q)$ to each party $\mathcal{P}_i$.*

publish$_1$: *Each party $\mathcal{P}_i$ chooses $s_i, e_i \leftarrow \chi$ as their secret key and error key, respectively, computes their public key $b_i = as_i + e_i \in R_q$, and sends their public key b_i to party $\mathcal{P}_{1-i}$.*

publish$_2$: *Party $\mathcal{P}_1$, upon receiving b_0 from $\mathcal{P}_0$, chooses a new error key $e_1' \leftarrow \chi$, computes $v = b_0 s_1 + e_1' \in R_q$, and uses the randomized doubling function on v to receive $\overline{v} \xleftarrow{R} \mathsf{dbl}(v) \in R_{2q}$. Using the cross-rounding function, $\mathcal{P}_1$ computes $c = \langle \overline{v} \rangle_{2q,2} \in \{0,1\}^m$ and sends c to $\mathcal{P}_0$.*

keygen: *In order to generate the shared key, party $\mathcal{P}_0$ uses the reconciliation function to output $k_{1,0} \leftarrow \mathsf{rec}(2b_1 s_0, c) \in \{0,1\}^m$. Party $\mathcal{P}_1$ simply computes $k_{0,1} = \lceil \overline{v} \rfloor_{2q,2} \in \{0,1\}^m$.*

Except with negligible probability $k_{0,1} = k_{1,0} = k$, i.e. this protocol satisfies correctness.

The security of the R-LWE KE protocol reduces to a decisional hardness problem dubbed the decision Diffie-Hellman-like (DDH-like) problem by Bos, Costello, Naehrig, Stebila [7]. For ease of proof later, we give a reformulated version of DDH-like problem, equivalent to the definition in [7, Definition 3].

Definition 37 (Decision Diffie-Hellman-like (DDH-like) problem)**.** *Let m, R, q, R_q, χ be D-R-LWE parameters. Given a tuple sampled uniformly from one of the following two distributions:*

- (a, b_0, b_1, c, κ), *where* $a \xleftarrow{R} R_q$, $s_0, s_1, e_0, e_1, e_1' \xleftarrow{R} \chi$, $b_i = as_i + e_i \in R_q$ *for* $i = 0, 1$, $v = b_0 s_1 + e_1'$, $\overline{v} \xleftarrow{R} \mathsf{dbl}(v)$, $c = \langle \overline{v} \rangle_{2q,2}$, *and* $\kappa = \lceil \overline{v} \rfloor_{2q,2} \in \{0,1\}^m$,
- (a, b_0, b_1, c, κ), *where* a, b_0, b_1, c *are as above and* $\kappa \xleftarrow{R} \{0,1\}^m$,

determine which distribution the tuple is sampled from.

Theorem 38 (Hardness of DDH-like problem; [7], theorem 1)**.** *Let m be a parameter, q an odd integer, and χ a distribution on R_q. If the D-R-LWE problem for m, R, q, R_q, χ is hard, then the DDH-like problem for m, R, q, R_q, χ is also hard.*

2.7 EVEN-MANSOUR

We state the Even-Mansour construction directly from [24], but using our notation, so our generalization will be clear.

Definition 39 (Even-Mansour (EM))**.** *Let $\{0,1\}^n$ denote the set of binary words of length n, let P be a common and publicly known permutation on $\{0,1\}^n$ and P^{-1} be its inverse. It is assumed that, for any given $x \in \{0,1\}^n$, it is easy to get $P(x)$ or $P^{-1}(x)$, either by a direct computation or by using an easily and commonly accessible black-box oracle.*

A key k consists of two subkeys, k_1 and k_2, each chosen at random from $\{0,1\}^n$. Initially, it is assumed that the key is known to the legitimate parties only; all other parties have no knowledge about it. Also, it is assumed that the key remains fixed and is used, by the legitimate parties, to encipher plaintexts or decipher ciphertexts, repeatedly, for a relatively long time.

The encryption $Enc_k(m)$ of a plaintext $m \in \{0,1\}^n$ by the key $k = (k_1, k_2)$, is performed by

$$Enc_k(m) = P(m \oplus k_1) \oplus k_2,$$

and the decryption of a ciphertext $c \in \{0,1\}^n$, is performed by

$$Dec_k(c) = P^{-1}(c \oplus k_2) \oplus k_1.$$

This scheme satisfies correctness.

It must be noted that the scheme is minimal, in the sense that having access to one of the keys, the other key can be found. Also, using only an inner key (or only an outer key) will allow basic cryptanalytic attacks.

It was shown by Kilian and Rogaway [42] that the above EM scheme is a pseudorandom permutation, i.e. for an adversary with only polynomially-many classical queries to an encryption oracle and the permutation oracles, it is indistinguishable from a random permutation. They only showed the pseudorandomness property, but state that their proof may be adapted to include a decryption oracle making the EM scheme a super pseudorandom permutation. They also note that these properties hold even for identical subkeys, that is, $k_1 = k_2$.

2.8 SECURITY MODELS

We present our standard security model in this section. We also define the stronger group-Canetti-Krawczyk plus (G-CK$^+$) model for authenticated group key exchanges (AGKEs). Although we only use the former security model in this dissertation, it is paramount for any student of modern GKEs to know what model should be employed in security proofs when possible, hence we present the latter model as well. We would also hope that our (authenticated) GKEs can be made to satisfy the latter security model, though at this time, we have not done so. The main difference between the two models is that the latter considers adversaries that may reveal the ephemeral keys and state information of honest parties. Our standard model is sometimes also called a weak-corruption model as opposed to G-CK$^+$, which is a strong-corruption model.

2.8.1 *Security model*

Consider a finite set of parties $\mathbb{P} = \{\mathcal{P}_0, \ldots, \mathcal{P}_\eta\}$ modelled by PPT Turing machines with security parameter 1^λ. A party $\mathcal{P}_i$ generates a pair of static keys: the static secret key SSK_i and the static public key SPK_i. For verification purposes, the public key is linked with $\mathcal{P}_i$'s identity in some trusted, public system like a public key infrastructure (PKI). In the same way, a party generates pair of ephemeral keys: the ephemeral secret key ESK_i and the ephemeral public EPK_i. In the case that the GKE does not employ any static keys, the keys are left blank.

2.8.1.1 *Session*

At any time, a subset $\{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_n}\} \subseteq \mathbb{P}$, where $2 \leq n \leq \eta$, can invoke a new GKE protocol. Such an invocation is called a *session*. A session is managed by a tuple $(\Pi, \mathcal{P}_{i_l}, \{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_n}\})$, where Π is the protocol

identifier and $\mathcal{P}_{i_l}$ is the party identifier.[11] For simplification of notation and without loss of generality, suppose that $\mathcal{P}_{i_l} = \mathcal{P}_l$. $\mathcal{P}_i$ outputs EPK_i to all relevant (necessary) parties and in turn receives $EPK_{i'}$ from $\mathcal{P}_{i'}$ from all relevant (necessary) parties.[12]

When $\mathcal{P}_i$ is the i-th party of a session, we may define the *session id* as the tuple $\mathsf{sid} = (\Pi, \mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \{EPK_1, \ldots, EPK_n\})$. We call $\mathcal{P}_i$ the *owner* of sid, if the second coordinate of sid is $\mathcal{P}_i$, and a *peer* of sid if it is not. A session is said to be *completed* if its owner computes the session key. We say that $(\Pi, \mathcal{P}_{i'}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \{EPK_1, \ldots, EPK_n\})$ is a *matching session* of $(\Pi, \mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \{EPK_1, \ldots, EPK_n\})$, where $i' \neq i$.

2.8.1.2 *Adversary*

Consider a (not necessarily classical) PPT Turing machine adversary $\mathcal{A}$ that controls all communication, including session activation and registration of parties. The adversary controls all communication through the following queries:

SEND$(\mathcal{P}_i, m)$: $\mathcal{P}_i$ is the receiver and the message has the form of $(\Pi, \mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \mathit{Init})$ if it is an initializing message for session activation, and includes $(\Pi, \mathcal{P}_{i'}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, EPK_{i'})$ with any other message.

ESTABLISH$(\mathcal{P}_i, SPK_i)$: Adds a new party to the total set of parties $\mathbb{P}$. In such a case, $\mathcal{A}$ is not required to supply or prove possession of a SSK_i corresponding to the linked SPK_i. If a party is registered by such a query, then the party is called *dishonest*, if not, the party is called *honest*.

Using the above queries, an adversary can start a session for any subset of parties.

The adversary is further given access to the following attack queries.

SESSIONREVEAL(sid): Reveals the session key of a session sid to the adversary, only if the session is completed.

STATICREVEAL$(\mathcal{P}_i)$: Reveals the static secret key of party $\mathcal{P}_i$ to $\mathcal{A}$.

11 Suzuki and Yoneyama [59] define their sessions with a 'role' for a party, which may be indexed differently from the party index, as well as a corresponding 'player' definition. In our protocols, the role is uniquely determined by the party index, which is uniquely determined by the placement in the double-tree (see Ch. 5). We therefore remove this 'role' (and 'player') from our definition of session.

12 Suzuki and Yoneyama [59] (and other GKE security models) assume that each party receives public keys from all other parties, but this forces all GKEs proven secure in this model to have at least linear complexity in n, we therefore alter the model slightly. What this means is that parties only need as many keys as are relevant/necessary to compute the session key.

2.8.1.3 *Security and freshness*

We require the definition of a fresh session in order to define our final security notion. Intuitively, freshness assures that the adversary has not queried any information that would allow it to trivially break scheme.

Definition 40. *Let* $\mathsf{sid}^* = (\Pi, \mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, EPK_i)$ *be a completed session between honest parties* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$*, owned by* $\mathcal{P}_i$*. If a matching session exists and we let* $\overline{\mathsf{sid}^*}$ *be such a matching session of* sid^**, then we call that* sid^* ***fresh*** *if none of the following is true:*

1. $\mathcal{A}$ *queried* SessionReveal(sid^*)*, or* SessionReveal$(\overline{\mathsf{sid}^*})$ *for any* $\overline{\mathsf{sid}^*}$ *if* $\overline{\mathsf{sid}^*}$ *exists; or*
2. $\overline{\mathsf{sid}^*}$ *exists, and* $\mathcal{A}$ *queried* StaticReveal$(\overline{\mathsf{sid}^*})$.

Otherwise, we call the session ***exposed****.*

To begin with, $\mathcal{A}$ is given a set of honest users and is allowed to make use of its attack queries as we described above. At some point, $\mathcal{A}$ makes the following query that will be used to determine whether it can break the scheme or not.

Test(sid^*): Issues the final test. At a point when the adversary decides that the data they have collected is enough, they may query the Test oracle for a challenge. First, a random bit b is generated. The adversary is then given the session key if $b = 1$, or instead a random key from the key space if $b = 0$.

The adversary is allowed to make adaptive queries before and after Test(sid^*) is issued, under the condition that it cannot expose the test session. Under the condition of freshness, the Test query requires that sid^* is completed. Eventually, $\mathcal{A}$ guesses a bit b'. We let $\mathsf{Succ}_{\mathcal{A}}(\Pi)$ be the event that $\mathcal{A}$ guesses $b' = b$, i.e. guesses the Test bit b correctly, and define the *advantage*

$$\mathsf{Adv}(\mathcal{A}) = \left| Pr[\mathsf{Succ}_{\mathcal{A}}(\Pi)] - \frac{1}{2} \right|.$$

In this model, we distinguish between a secure GKE and a secure AGKE. The difference is that in the GKE security, we consider only *passive* adversaries that, other than using the Establish query, does not create its own messages, honestly relaying messages between other parties and following the protocol prescription. For AGKE security, we consider *malicious* adversaries that have no such limits, freely able to create, omit, and resend messages.

Definition 41. *A group key exchange is said to be a* ***secure GKE*** *if for any passive PPT adversary* $\mathcal{A}$*,*

1. *If two honest parties complete matching sessions, these sessions produce the same session key as output, except with at most negligible probability;*

2. $Adv(\mathcal{A})$ *is negligible in security parameter* 1^λ *for the test session* sid^*.

Definition 42. *A group key exchange is said to be a* ***secure AGKE*** *if for any malicious PPT adversary* $\mathcal{A}$*,*

1. *If two honest parties complete matching sessions, these sessions produce the same session key as output, except with at most negligible probability;*

2. $Adv(\mathcal{A})$ *is negligible in security parameter* 1^λ *for the test session* sid^*.

The StaticReveal query is considered for the sake of forward security, i.e. revealing static keys does not expose previous session keys. Hence, if a GKE is secure in our model, it can be said to have *forward security* or be *forward-secure*.

2.8.2 *G-CK$^+$ security model*

We now define the extra requirements that make up the G-CK$^+$ model of Suzuki and Yoneyama [59]. The model is the same as the model given in the preceding section but the adversary is given access to two more attack queries, StateReveal and EphemeralReveal, which further complicates the freshness definition.

These queries change the security model into a model that considers active insiders, i.e. an adversary can learn even session-specific secret information. We give the full scope of attack queries and freshness requirements here for easy perusal but omit the other elements of the security model as they are identical to the definitions given in the previous subsection.

Along with the Send and Establish queries (see p. 31), the adversary is given access to the following attack queries, quoted from [59].

SessionReveal(sid): Reveals the session key of a session sid to the adversary, only if the session is completed.

StateReveal(sid): Reveals to the adversary the session state of the owner of the sid if the session is not yet completed, i.e. the session key has yet to be established. The session state contains all ephemeral secret keys and intermediate computation results except for immediately erased information, but it does not contain the static secret key. The protocol specifies what the session state contains.

StaticReveal($\mathcal{P}_i$): Reveals the static secret key of party $\mathcal{P}_i$ to $\mathcal{A}$.

EphemeralReveal(sid): Reveals to the adversary all ephemeral secret keys of the owner of sid if the session is not yet completed. This does not reveal other state information such that an adversary might trivially win.

Definition 43 ([59]). *Let* $\mathsf{sid}^* = (\Pi, \mathcal{P}_i, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, EPK_i)$ *be a completed session between honest parties* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$*, owned by* $\mathcal{P}_i$*. If a matching session exists, then let* $\overline{\mathsf{sid}^*}$ *be a matching session of* sid^**. We say that* sid^* *is* ***fresh*** *if none of the following is true:*

1. $\mathcal{A}$ *queried* SessionReveal(sid^*)*, or* SessionReveal($\overline{\mathsf{sid}^*}$) *for any* $\overline{\mathsf{sid}^*}$ *if* $\overline{\mathsf{sid}^*}$ *exists; or*
2. $\overline{\mathsf{sid}^*}$ *exists, and* $\mathcal{A}$ *queried* StateReveal(sid^*) *or* StaticReveal($\overline{\mathsf{sid}^*}$)*; or*
3. $\overline{\mathsf{sid}^*}$ *does not exist, and* $\mathcal{A}$ *queried* StateReveal(sid^*)*; or*
4. $\mathcal{A}$ *queried both* StaticReveal($\mathcal{P}_i$) *and* EphemeralReveal(sid^*)*; or*
5. $\overline{\mathsf{sid}^*}$ *exists (the owner of* $\overline{\mathsf{sid}^*}$ *is* $\mathcal{P}_{i'}$*), and* $\mathcal{A}$ *queried both* StaticReveal($\mathcal{P}_{i'}$) *and the query* EphemeralReveal($\overline{\mathsf{sid}^*}$)*; or*
6. $\overline{\mathsf{sid}^*}$ *does not exist, and* $\mathcal{A}$ *queried* StaticReveal($\mathcal{P}_{i'}$) *for any intended peer* $\mathcal{P}_{i'}$ *of* $\mathcal{P}_i$*.*

Otherwise, we call the session ***exposed****.*

2.8.3 *Security model discussion*

Manulis et al. [49] and Suzuki and Yoneyama [59] both consider malicious, adaptive adversaries, the latter being the G-CK$^+$ model. Their models offer security by considering leakage of state information or ephemeral secrets in the test session itself. These are therefore very strong models - G-CK$^+$ being the strongest - giving the adversary practically as much information as they could possibly get about a GKE session through their attack oracles. Unfortunately, giving the adversary even a single ephemeral key in either BDI or BDII allows an adversary the ability to distinguish a GKE session key from random as it can compute the session key by using the ephemeral key and all the public information.

The corresponding GKEs that [49] and [59] propose are secure in their respective security models but the other PQC GKEs that we reference in this dissertation prove security in the same, weaker model that we do. Nonetheless, we define the strongest of the (authenticated) GKE security models, G-CK$^+$, here for future work. See Sect. 7.3 for some thoughts on this future work.

Part II

RESEARCH

3

GROUP KEY EXCHANGES FROM ISOGENIES

3.1 INTRODUCTION

Over the decades, the Diffie-Hellman key exchange protocol has been used in copious systems to securely exchange keys between two parties. In today's modern environment, simple two-party KEs do not always suffice as online services include interactions between multiple parties all at once. As a leading example, consider that many instances of social media allow several persons to make group chats. The security of those group chats require a single shared key between all the group members but that presents the problem of how to create such a key. The process of making such a shared key for a group of parties is what we call a group key exchange (GKE).

We might trivially make an n party GKE using $n-1$ rounds of two-party KEs. Such a trivial GKE will have $O(n)$ communication and memory complexity. Here, the communication complexity is the maximal number of messages any singular party receives and memory complexity is the maximal number of values any singular party needs to store for the session key computation. However, it would be ideal if we could create a GKE that has fewer rounds and lower communication and memory complexity, especially for larger groups. Burmester and Desmedt [8] created the GKE we call BDI (see Sect. 2.4). Although this n-party GKE also has linear communication and memory complexity, it does reduce the number of rounds to 2.

The security of BDI relies on the security of the underlying two-party KE protocol, Diffie-Hellman key exchange. Unfortunately for Burmester and Desmedt, the emergence of quantum computers presents a real and pressing threat to Diffie-Hellman-based constructions. DH is based on the classically hard mathematical problem called the discrete logarithm problem. It turns out that there exist quantum algorithms for solving exactly this problem in such a way that protocols that rely on it can no longer be considered secure (against quantum adversaries), which in turn means that also DH, and thereby BDI, can no longer be considered secure (against quantum adversaries).

Considering the problem of quantum security and the ongoing need for key exchanges, Jao and De Feo [38] proposed SIDH, a drop-in candidate for post-quantum KE. This KE relies on a relatively new problem for security, namely the indistinguishability of the SSDDH problem. This problem relies on the difficult task of solving what is called the isogeny finding problem, a problem which is thought to be hard even for quantum computers.

Naturally an isogeny based GKE would also emerge. Furukawa, Kunihiro, and Takashima [27] proposed supersingular isogeny Burmester-Desmedt (SIBD), a generalization of BDI that uses SIDH as the underlying KE instead of DH. The SIBD protocol arranges parties in a ring-like structure that has each party doing equal amounts of computation in order to compute the shared key, which is a product of j-invariants. Being based on BDI, SIBD is straddled with the same $O(n)$ communication and memory complexity.

Burmester and Desmedt [9] proposed yet another GKE protocol that we call BDII (see Sect. 2.4). BDII is also a n-party GKE that completes in only 2-rounds, but unlike BDI is tree-based. This gives BDII communication and memory complexity $O(\log_2 n)$. Furthermore, the tree structure allows parties to be arranged according to processing power and memory capability, such that BDII is applicable in unbalanced networks.

Even though the isogeny based generalization of BDI exists, generalizing BDII using SIDH and proving its post-quantum security, is actually non-trivial. First of all, two parties in SIDH sharing a basis has public values that allow the secret keys of either party to be found. This can be done because it requires solving the elliptic curve analogue of the discrete logarithm problem, which, unfortunately, is an easy problem for a quantum adversary, in polynomial-time even. Second of all, consider that for DH, the group elements involved possess associativity and transitivity, allowing the group shared key to have the same form as a DH key: g^x for some generator $g \in \mathbb{Z}_p$ and $x \in \mathbb{Z}_p^*$, where p is a prime. For SIDH, on the other hand, the shared key is the j-invariant of an elliptic curve (or just an elliptic curve). The set of elliptic curve does not form a group in the majority of cases and there is no obvious way to combine elliptic curves. Furthermore, if the elliptic curves are supersingular, then a product or sum of j-invariants is not likely to be a j-invariant. The first problem is here circumvented by carefully indexing parties and defining a 'score' function. The second problem is circumvented by reducing our GKE protocol to the SSDDH problem, giving our protocol the same security as SIDH. This furthermore avoids defining a new hard problem as supersingular isogeny Burmester-Desmedt (SIBD) must.[1]

We call our generalization supersingular isogeny tree-based GKE (SIT). The shared key in SIT is the j-invariant of only a single SIDH KE between two initial parties. SIT is a 2-round, is post-quantum secure, and when each (non-leaf) party in the tree is assumed to have l children, has communication and memory complexity $O(\log_l n)$.

A GKE alone is not enough in the real world as there may exist malicious outsiders able to influence a communication network as

1 The security proof SIBD given by Furukawa, Kunihiro, and Takashima [27] is unfortunately faulty. Takashima [60] defines a new hard problem and supplies a corresponding proof to remedy this.

they see fit. For this reason an authenticated GKE protocol is necessary. Although specialized AGKEs exist, Katz and Yung [41] create a generic compiler that employs a secure signature protocol and turns any GKE protocol into an authenticated one. Unfortunately, the compiler results in linear communication complexity even when the underlying GKE has lower communication complexity. We therefore introduce a new compiler that maintains whichever communication complexity the underlying GKE achieves. For supersingular isogeny tree-based GKE (SIT), we call the compiled AGKE for authenticated SIT (A-SIT).

3.2 RELATED WORK

Azarderakhsh, Jalali, Jao, and Soukharev [4] attempted to create the first non-trivial isogeny based GKE, Isog-GKE. Their idea was to arrange the parties sequentially and have each party compute isogenies on all the previous parties' public curves. This was repeated in the opposite direction as well. At the end of the procedure, each party would hold an elliptic curve isogenous to each of the other parties' elliptic curve. They did exceed in creating a 2 round GKE but the resulting scheme required $O(n)$ isogenies and the same amount of public values, communicated values, and stored values. Isogeny computation is rather slow in terms of computation time, so the fewer isogeny computations, the better. Unfortunately, their scheme has recently been completely broken by de Quehen et al. [55] using an improved torsion-point application on the adaptive attack of Galbraith et al. [28]. Their work is therefore better as an example of what has been tried and what will not work for isogeny based GKEs.

The emergence of quantum computers will threaten the security of DH based constructions such as BDI, as we mentioned in our introduction. In order to allay such concerns, [27] brilliantly generalized the BDI construction relying on SIDH instead of DH, where a product of j-invariants then replaces a group element as the shared secret. As we use the framework of supersingular isogeny Burmester-Desmedt (SIBD) in our own protocol we first give the construction below.

Suppose first that the parties wishing to do a group key exchange are indexed by $1, \ldots, n$ where n is even.[2] Party indexes are taken modulo n, e.g. $\mathcal{P}_0 := \mathcal{P}_n$ and $\mathcal{P}_{n+1} := \mathcal{P}_1$. We must also introduce the map $\iota = \iota(i) := i \mod 2$. The public parameter generating algorithm GenSS is as in SIDH (Protocol 23).

Protocol 44 (SIBD [27])**.** *The SIBD protocol is as follows*

setup: *Takes security parameter* 1^λ *and the number of parties* n *as inputs. In the same way as for SIDH, the* GenSS *algorithm outputs:*

$$\mathfrak{P} := (p, E, \{P_0, Q_0\}, \{P_1, Q_1\}) \xleftarrow{R} \mathsf{GenSS}(1^\lambda).$$

2 Although the authors claim their scheme generalizes to odd n, their indexing method does not allow for such odd n.

publish$_1$: *Takes the party index i and the public parameter tuple $\mathfrak{P}$ as inputs.*

1. *Party $\mathcal{P}_i$ first chooses $r_i \xleftarrow{R} \mathbb{Z}/\ell_\iota^{e_\iota}\mathbb{Z}$ uniformly at random then computes $R_i := P_\iota + [r_i]Q_\iota$.*
2. *Party $\mathcal{P}_i$ then computes the isogeny ϕ_i and elliptic curve $E_{R_i} \cong E/\langle R_i\rangle$ s.t. $\phi_i : E \to E_{R_i}$, where $\ker(\phi_i) = \langle R_i\rangle$.*
3. *Party $\mathcal{P}_i$ then sets*

$$sk_i^1 = r_i \text{ and } pk_i^1 = (E_{R_i}, \phi_i(P_{1-\iota}), \phi_i(Q_{1-\iota})).$$

4. *Party $\mathcal{P}_i$ broadcasts pk_i^1 to all the other users.*

publish$_2$: *Takes the party index i,* params, *the secret key sk_i^1, and the public keys $\{pk_{i-1}^1, pk_{i+1}^1\}$.*

1. *Party $\mathcal{P}_i$ executes SIDH with the neighbouring parties $\mathcal{P}_{i-1}$ and $\mathcal{P}_{i+1}$ to obtain j-invariants $j(E_{R_{i-1},R_i})$ and $j(E_{R_i,R_{i+1}})$.*
2. *$\mathcal{P}_i$ then computes $u_i := j(E_{R_{i-1},R_i}) \cdot j(E_{R_i,R_{i+1}})^{-1}$ and sets $pk_i^2 := u_i$.*
3. *Party $\mathcal{P}_i$ broadcasts pk_i^2 to all the other parties.*

keygen: *Party $\mathcal{P}_i$ collects all the $\{pk_i^2\}_{i=1,\dots,n}$ and uses its secret key sk_i^1 to compute*

$$K_i = j(E_{R_{i-1},R_i})^n \cdot u_i^{n-1} \cdot u_{i+1}^{n-2} \cdot \cdots \cdot u_{i-3}^2 \cdot u_{i-2}.$$

We can easily verify that

$$K_i = j(E_{R_1,R_2}) \cdots j(E_{R_n,R_1}) =: K$$

holds for any $i \in \{1, \dots, n\}$.

We draw attention to the fact that the shared key is a product of the j-invariants, indeed of *all* the pairwise shared elliptic curve j-invariants.

3.3 SUPERSINGULAR ISOGENY TREE-BASED GKE (SIT)

Assume that from a set of parties, $\mathbb{P}$, we have a subset of $n \geq 2$ parties $\{\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_{n-1}\}$, re-indexing if need be, that wish to generate a shared key. For supersingular isogeny tree-based GKE (SIT), like BDII, the parties are assumed to be arranged in two trees, connected at the roots by parties $\mathcal{P}_0$ and $\mathcal{P}_1$, and arranging the parties in ascending order from the top-leftmost root, going right, and continuing down the tree level-wise, not branch-wise (see Figure 3.1). We choose to call this a **double-tree**. We assume that all parties are unique, i.e. a party appears at most once in the tree.

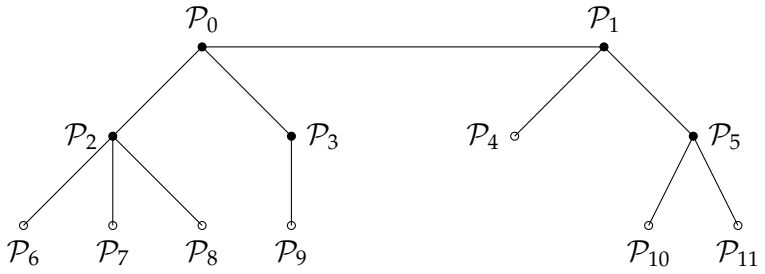

Figure 3.1: Possible SIT graph configuration for $n = 12$

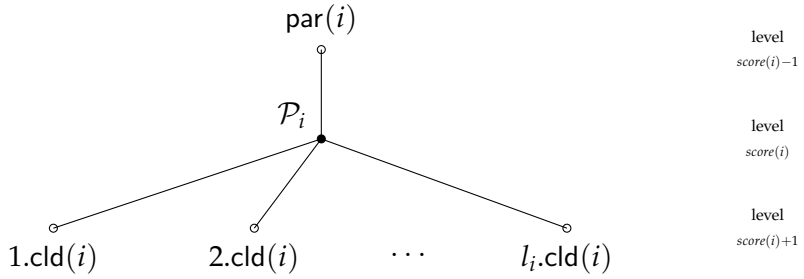

Figure 3.2: The neighbours of $\mathcal{P}_i$: Parent and children

As shown for BDII [9], we can let there be a variable number of children per node (see Figure 3.1), however, our notation and definitions vary from those given in [9]. Excepting the leaves of the tree, each party $\mathcal{P}_i$ has a parent $\mathsf{par}(i)$, and a set of children $\mathfrak{l}.\mathsf{cld}(i)$ for $\mathfrak{l} = 1, 2, \ldots, l_i$ where $0 \leq l_i \leq n-2$ is the number of children of $\mathcal{P}_i$, that are all considered the *neighbours* of $\mathcal{P}_i$ (see Figure 3.2). For all $\mathcal{P}_i$ that are leaves, $l_i = 0$. We let $\mathsf{ancestors}(i)$ be the set of indexes of all ancestors of a party $\mathcal{P}_i$, including i but having removed 0 and 1. $\mathcal{P}_0$ and $\mathcal{P}_1$ are assumed to be parents of each other.

We define **score**[3] recursively in the following way: $score(0) := 0, score(1) := 1, score(i) = score(\mathsf{par}(i)) + 1$, where $i \geq 2$ is the index of the party $\mathcal{P}_i$. We assume that each party calculates their own score when the graph is fixed and does not occur as a bandwidth or computational cost in our protocol. Using score, we define the map $\iota = \iota(i) := score(i) \mod 2$.

For the remainder of this dissertation, we differentiate between broadcasting and multicasting in that ***broadcasting*** is data transmission to all parties involved in a GKE, while ***multicasting*** is data transmission to a subset of the parties: at most the parent and all descendants. For use in our authenticated GKE compiler in Sect. 3.5, we add a *session name* to the GKE protocol parameter generation. The session name is a unique name for each instantiation of the protocol.

3 Although the original BDII scheme has no such concept, we introduce a 'score' for each party, needed in order to alternate SIDH bases so as to avoid the attack outlined in Note 29.

We assume there exists a set of all parties $\mathbb{P}$. Consider the following protocol. Given a security parameter 1^λ and a set of $n \leq |\mathbb{P}|$ parties, Gen_{SIT} outputs the SIDH-based tuple $(p, E, \{P_0, Q_0\}, \{P_1, Q_1\}, \Gamma, sID)$, where $p = f\ell_0^{e_0}\ell_1^{e_1} \pm 1$ is prime for a small integer $f > 0$ and $\ell_0^{e_0} \approx \ell_1^{e_1}$ (usually $\ell_0 = 2$ and $\ell_1 = 3$), E is a randomly chosen supersingular elliptic curve over $\mathbb{F}_{p^2}$ such that $\#E(\mathbb{F}_{p^2}) = (p \pm 1)^2$, $\{P_k, Q_k\}$ is a randomly chosen basis of $E[\ell_k^{e_k}]$ for $k \in \{0, 1\}$, Γ is a double-tree for n parties, and sN is a unique session name.

Protocol 45 (Supersingular isogeny tree-based GKE (SIT))**.** *The protocol is as following.*

setup: *For a security parameter 1^λ and the number of users n, the algorithm outputs to each party $\mathcal{P}_i$ the tuple:*

$$\mathfrak{P} := (p, E, \{P_0, Q_0\}, \{P_1, Q_1\}, \Gamma, sN) \xleftarrow{R} \mathsf{Gen}_{SIT}(1^\lambda, n),$$

where sN is the unique session name.

publish$_1$: *Given $\mathfrak{P}$, individually, each $\mathcal{P}_i$ computes*

$$Z_i = (E_i, \phi_i(P_{1-\iota}), \phi_i(Q_{1-\iota})),$$

where $\phi_i : E \to E_i \cong E/\langle R_i \rangle$ with $R_i := P_\iota + [r_i]Q_\iota$ for a secret $r_i \xleftarrow{R} \mathbb{Z}/\ell_\iota^{e_\iota}\mathbb{Z}$ chosen uniformly at random. Z_i is $\mathcal{P}_i$'s public key and r_i is its secret key. $\mathcal{P}_i$ then multicasts Z_i it to its neighbours (parent and l_i children), along with its identification pair $(\mathcal{P}_i, sN)$. Each party $\mathcal{P}_i$ then computes $l_i + 1$ SIDH shared keys (or 1 in the case of leaves, as they have no children) using the neighbours' public keys:

$$\begin{aligned} E_{\mathsf{par}(i),i} &= \phi_i^{\mathsf{par}(i)}(E_{\mathsf{par}(i)}) \\ &\cong E/\langle P_\iota + [r_i]Q_\iota, P_{1-\iota} + [r_{\mathsf{par}(i)}]Q_{1-\iota}\rangle, \\ E_{\mathfrak{l}.\mathsf{cld}(i),i} &= \phi_i^{\mathfrak{l}.\mathsf{cld}(i)}(E_{\mathfrak{l}.\mathsf{cld}(i)}) \\ &\cong E/\langle P_\iota + [r_i]Q_\iota, P_{1-\iota} + [r_{\mathfrak{l}.\mathsf{cld}(i)}]Q_{1-\iota}\rangle, \end{aligned}$$

for each $\mathfrak{l} \in \{1, \ldots, l_i\}$, where

$$\begin{aligned} \ker(\phi_i^{\mathsf{par}(i)}) &= \langle \phi_{\mathsf{par}(i)}(P_\iota) + [r_i]\phi_{\mathsf{par}(i)}(Q_\iota)\rangle, \\ \ker(\phi_i^{\mathfrak{l}.\mathsf{cld}(i)}) &= \langle \phi_{\mathfrak{l}.\mathsf{cld}(i)}(P_\iota) + [r_i]\phi_{\mathfrak{l}.\mathsf{cld}(i)}(Q_\iota)\rangle. \end{aligned}$$

publish$_2$: *Each $\mathcal{P}_i$ with children computes*

$$x_{\mathfrak{l}.\mathsf{cld}(i)} = j(E_{\mathsf{par}(i),i}) - j(E_{\mathfrak{l}.\mathsf{cld}(i),i}),$$

and multicasts this value to the respective descendants, for each $\mathfrak{l} \in \{1, \ldots, l_i\}$.

keygen: *Each $\mathcal{P}_i$ computes a shared key*

$$K_i = j(E_{\mathsf{par}(i),i}) + \sum_{m \in \mathsf{ancestors}(i)} x_m = j(E_{0,1}) = K.$$

Note 46. *Once the setup algorithm establishes the double-tree graph configuration we assume that it is both fixed for the session and also publicly available to all parties (and adversaries). The means that for each protocol execution, the setup algorithm must be run anew. We then assume that each party checks the tree to see their own position as well their neighbours, descendants, and ancestors.*

Proposition 47 (SIT correctness)**.** *Each party in SIT computes the same key $K = j(E_{0,1})$, i.e. the session key of the group is simply the shared key of the initial parties $\mathcal{P}_0$ and $\mathcal{P}_1$.*

Proof. Proof by induction. Obviously, $K_0 = K_1 = j(E_{0,1})$. Assume that $K_{\mathsf{par}(i)} = K$ then, as

$$K_{\mathsf{par}(i)} = j(E_{\mathsf{par}(\mathsf{par}(i)),\mathsf{par}(i)}) + \sum_{m \in \mathsf{ancestors}(\mathsf{par}(i))} x_m,$$

we have that

$$\begin{aligned} K_i &= j(E_{\mathsf{par}(i),i}) + \sum_{m \in \mathsf{ancestors}(i)} x_m \\ &= j(E_{\mathsf{par}(i),i}) + \Big(j(E_{\mathsf{par}(\mathsf{par}(i)),\mathsf{par}(i)}) - j(E_{i,\mathsf{par}(i)})\Big) \\ &\qquad + \sum_{m \in \mathsf{ancestors}(\mathsf{par}(i))} x_m \\ &= K_{\mathsf{par}(i)} = K. \end{aligned}$$

□

Note 48. *By Silverman [57] (V.4.4), if $p \equiv 1 \pmod 3$, then there are no supersingular elliptic curves with j-invariant equal to 0, but if $p \equiv 2 \pmod 3$, then there exists at least one supersingular curve with j-invariant equal to 0. If we use the product operation in the definition of the x-values and the session key computation, some x-values might equal 0 forcing the session key to be incalculable for some parties. To avoid this unfortunate situation, we define SIT using the addition operation in $\mathbb{F}_q$ on the j-invariants used to calculate x-values and session key, instead of the product as done in SIBD. As addition is a more efficient computation this choice seems ideal and urge implementers to do so.*

Theorem 49. *Assuming the SSDDH problem is (post-quantum) hard, the SIT protocol given in Protocol 45 is a (post-quantum) secure GKE (Definition 41), with forward security.*

Proof. We must show that Protocol 45 satisfies the security notion given in Definition 41. The first requirement is satisfied by the correctness shown in Proposition 47.

For the second requirement, assume that there exists a (not necessarily classical) polynomial-time adversary $\mathcal{A}$ with non-negligible

Algorithm 1 SSDDH distinguisher, $\mathcal{D}$.

Input: $E_0, E_1, \phi_0(P_1), \phi_0(Q_1), \phi_1(P_0), \phi_1(Q_0), E'$

1: $h \xleftarrow{R} \{1, \ldots, \Lambda\}$, where Λ is an upper bound on the number of sessions activated by $\mathcal{A}$ in any interaction.
2: Invoke $\mathcal{A}$ and simulate protocol to $\mathcal{A}$, except for the h-th activated protocol session.
3: For the h-th session:
4: Set $\mathfrak{P} := (p, E, \{P_0, Q_0\}, \{P_1, Q_1\}, \Gamma, sN)$, where Γ is an n-party binary graph and sN is the session name.
5: Set $Z'_0 = (E_0, \phi_0(P_1), \phi_0(Q_1))$ and $Z'_1 = (E_1, \phi_1(P_0), \phi_1(Q_0))$. Choose $r_i \xleftarrow{R} \mathbb{Z}/\ell_i^{e_i}\mathbb{Z}$ for $i = 2, \ldots, n-1$ and set $Z'_i = (E_i (\cong E/\langle R_i \rangle), \phi_i(P_{1-\iota}), \phi_i(Q_{1-\iota}))$. Simulate multicasting for each $\mathcal{P}_i$ along with identifying information $(\mathcal{P}_i, s)$.
6: Set

$$x'_{\mathfrak{l}.\mathsf{cld}(0)} := j(E') - j(E_{0,\mathfrak{l}.\mathsf{cld}(0)}), \forall \mathfrak{l} \in \{1, 2, \ldots, l_0\},$$
$$x'_{\mathfrak{l}.\mathsf{cld}(1)} := j(E') - j(E_{1,\mathfrak{l}.\mathsf{cld}(1)}), \forall \mathfrak{l} \in \{1, 2, \ldots, l_1\}, \text{ and}$$
$$x'_{\mathfrak{l}.\mathsf{cld}(i)} := j(E_{\mathsf{par}(i),i}) - j(E_{\mathfrak{l}.\mathsf{cld}(i),i}), \forall \mathfrak{l} \in \{1, 2, \ldots, l_i\},$$

for $i \geq 2$ where $\mathcal{P}_i$ is not a leaf in Γ.
7: **if** the h-th session is chosen by $\mathcal{A}$ as the test session **then**
8: Provide $\mathcal{A}$ as the answer to the distinguishing query, i.e.
9: $d \leftarrow \mathcal{A}$'s output
10: **else**
11: $d \xleftarrow{R} \{0, 1\}$.

Output: d

advantage $\mathsf{Adv}(\mathcal{A}) = \varepsilon$. We build a polynomial-time distinguisher $\mathcal{D}$ for the SSDDH problem in Algorithm 1.

As an analysis of our distinguishing algorithm, we note the following. For the h-th session, using the public information for $\mathcal{P}_0$, namely $\phi_0(P_1), \phi_0(Q_1)$, it does SIDH key exchange with the secret key $r_{\mathfrak{l}.\mathsf{cld}(0)}$ of party $\mathcal{P}_{\mathfrak{l}.\mathsf{cld}(0)}$, giving the curve $\phi_{\mathfrak{l}.\mathsf{cld}(0)}(E_0) = E_{0,\mathfrak{l}.\mathsf{cld}(0)} \cong E_{\mathfrak{l}.\mathsf{cld}(0),0}$, for each $\mathfrak{l} \in \{1, \ldots, l_0\}$ where l_0 is the number of children for $\mathcal{P}_0$ in Γ. Likewise, using the public information for $\mathcal{P}_1$, it does SIDH key exchange with the secret key $r_{\mathfrak{l}.\mathsf{cld}(1)}$ of party $\mathcal{P}_{\mathfrak{l}.\mathsf{cld}(1)}$, finding $E_{1,\mathfrak{l}.\mathsf{cld}(1)} \cong E_{\mathfrak{l}.\mathsf{cld}(1),1}$, for each $\mathfrak{l} \in \{1, \ldots, l_1\}$ where l_1 is the number of children for $\mathcal{P}_1$ in Γ. All other curves may be computed as the secret keys for $\mathcal{P}_i$ are known for $i = 2, \ldots, n-1$.[4]

As the r_i are chosen uniformly at random for $i \geq 2$, the distribution of the Z'_i and $x'_{\mathfrak{l}.\mathsf{cld}(i)}$ are identical to that in an SIT instance.

4 $\mathcal{D}$ needs at most $(2n-4)t_{isog}$ additional time in order to generate all the necessary isogenies for the transcript, where t_{isog} is the amount of time needed to do a single SIDH isogeny computation.

The transcript given to $\mathcal{A}$ by $\mathcal{D}$ is

$$(Z'_0, \ldots, Z'_{n-1}, x'_{1.\mathsf{cld}(0)}, x'_{2.\mathsf{cld}(0)}, \ldots, x'_{l_0.\mathsf{cld}(0)}, \\ x'_{1.\mathsf{cld}(1)}, \ldots, x'_{(l_{n-1}).\mathsf{cld}(n-1)}),$$

where we assign a blank value for the x' value when there is no child.

If the h-th session is $\mathcal{A}$'s test session, then $\mathcal{D}$ is issuing $K = j(E')$.

If K is a valid SIT key, then $j(E') = K = j(E_{0,1}) = j(E_{1,0})$, in other words, the tuple

$$(E_0, E_1, \phi_0(P_1), \phi_0(Q_1), \phi_1(P_0), \phi_1(Q_0), E')$$

where $E' \cong E_{0,1} \cong E_{1,0}$, is a tuple from the first distribution in the SSDDH problem (Definition 27). If K is not a valid SIT key, then it is a tuple from the second distribution in the SSDDH problem (Definition 27).

If the test session is *not* the h-th session, then $\mathcal{D}$ outputs a random bit, i.e. it has advantage 0. If the test session *is* the h-th session, which happens with probability $1/\Lambda$, then $\mathcal{A}$ will succeed with advantage ε. Hence, the final advantage of the SSDDH distinguisher $\mathcal{D}$ is ε/Λ, which is non-negligible.

This protocol does not involve long-term secrets, all ephemeral keys being generated anew in each session such that ephemeral keys generated in one session are independent of those in another session, so too the session key. Hence, an adversary having access to keys from one session, will not be able to use their knowledge to learn the session key from any previous session. □

3.3.1 *CSIDH version*

Although we will not get into the details of the Castryck et al. commutative supersingular isogeny Diffie-Hellman key exchange [11] - another isogeny based PQC key exchange that uses group actions - it will be illustrative to see both how group actions work and how the SIT GKE could look for other isogeny based KEs.

For our purposes, we simply define the CSIDH group action to be a group acting on a set of elliptic curves with the notation, $[\mathfrak{a}] \star E$. The action is regular and, in particular, it is the action of the ideal class group $cl(\mathcal{O})$, for an imaginary quadratic order $\mathcal{O}$, acting on the set of supersingular elliptic curves over $\mathbb{F}_p$ with endomorphism ring isomorphic to the order (see [11] for further details). All we really need to know for this subsection is that the group action is commutative, i.e. $[\mathfrak{ab}] \star E = [\mathfrak{ba}] \star E$.

The KE that can be built from this group action is exactly like the DH KE described in Section 2.3.1, with public keys being actions $[\mathfrak{x}_i] \star E$ for the secret keys $[\mathfrak{x}_i]$ for $i \in \{0, 1\}$ and a fixed elliptic curve E. The shared key is $[\mathfrak{x}_0\mathfrak{x}_1] \star E$.

The CSIDH GKE would first fix the relevant CSIDH parameters, including an elliptic curve E, as public parameters, along with the graph and unique session name. There is no need for picking bases as CSIDH is a non-interactive key exchange (for the distinction, see Chapter 5 below). Each party would then pick their secret keys $\mathfrak{x}_i$ uniformly at random and do pairwise CSIDH KEs with their parent and children using the respective public keys $[\mathfrak{x}_i] \star E$.

Each party should now have the following shared keys:

$$E_{\mathsf{par}(i),i} = [\mathfrak{x}_{\mathsf{par}(i)}\mathfrak{x}_i] \star E, \text{ and } E_{\mathfrak{l}.\mathsf{cld}(i),i} = [\mathfrak{x}_{\mathsf{cld}(i)}\mathfrak{x}_i] \star E,$$

for all $\mathfrak{l} \in \{0, 1, \ldots, l_i\}$. Each party then proceeds to compute

$$x_{\mathfrak{l}.\mathsf{cld}(i)} = j(E_{\mathsf{par}(i),i}) - j(E_{\mathfrak{l}.\mathsf{cld}(i),i}),$$

for each $\mathfrak{l} \in \{1, \ldots, l_i\}$, and multicast the respective value to the respective descendants, just as in the $\mathbf{publish_2}$ step of the SIT protocol. Each party may then compute the shared key in the same way, namely

$$K_i = j(E_{\mathsf{par}(i),i}) + \sum_{m \in \mathsf{ancestors}(i)} x_m = j(E_{0,1}) = j([\mathfrak{x}_0\mathfrak{x}_1] \star E).$$

3.4 PEER-TO-PEER SIT (P2P-SIT)

We also give a peer-to-peer version of our SIT protocol, as done by Desmedt, Lange, and Burmester [18], and call this protocol the peer-to-peer SIT (P2P-SIT) protocol. For the peer-to-peer version, the number of rounds and the communication complexity switches. This means that the communication complexity becomes constant at the same time that the number of rounds becomes logarithmic. The protocol differences may be found after step $\mathbf{publish_1}$.

Here, we again consider a double-tree for n-parties, Γ, and use the term "multicast" to mean that a party only sends a message to a discrete subset of all potential parties: at most its descendants and parent.

We assume there exists a set of all parties $\mathbb{P}$. Given a security parameter 1^λ and a set of $n \leq |\mathbb{P}|$ parties, Gen_{SIT} outputs the SIDH-based tuple $(p, E, \{P_0, Q_0\}, \{P_1, Q_1\}, \Gamma, sN)$, where $p = f\ell_0^{e_0}\ell_1^{e_1} \pm 1$ is prime for a small integer $f > 0$ and $\ell_0^{e_0} \approx \ell_1^{e_1}$ (usually $\ell_0 = 2$ and $\ell_1 = 3$), E is a randomly chosen supersingular elliptic curve over $\mathbb{F}_{p^2}$ such that $\#E(\mathbb{F}_{p^2}) = (p \pm 1)^2$, $\{P_k, Q_k\}$ is a randomly chosen basis of $E[\ell_k^{e_k}]$ for $k \in \{0, 1\}$, Γ is a double-tree for n parties, and sN is a unique session name.

Protocol 50 (peer-to-peer SIT (P2P-SIT))**.** *The protocol is as follows.*

setup: *For security parameter 1^λ and the number of users n, the public parameter generating algorithm Gen_{SIT} outputs to each party $\mathcal{P}_i$ the tuple:*

$$\mathfrak{P} := (p, E, \{P_0, Q_0\}, \{P_1, Q_1\}, \Gamma, sN) \xleftarrow{R} \mathsf{Gen}_{SIT}(1^\lambda, n),$$

where Γ *includes the party arrangement, i.e. ancestors and descendants.*

publish$_1$: *Given* $(p, E, \{P_0, Q_0\}, \{P_1, Q_1\}, \Gamma, sN)$*, individually, each* $\mathcal{P}_i$ *computes*

$$Z_i = (E_i, \phi_i(P_{1-\iota}), \phi_i(Q_{1-\iota})),$$

where $\phi_i : E \to E_i \cong E/\langle R_i \rangle$ *with* $R_i := P_\iota + [r_i]Q_\iota$ *for a randomly chosen, secret* $r_i \xleftarrow{R} \mathbb{Z}/\ell_\iota^{e_\iota}\mathbb{Z}$*.* Z_i *is* $\mathcal{P}_i$*'s public key and* r_i *is its secret key.* $\mathcal{P}_i$ *then multicasts* Z_i *it to its neighbours (parent and* l_i *children), along with its identification pair* $(\mathcal{P}_i, sN)$*. Each party* $\mathcal{P}_i$ *then computes* $l_i + 1$ *SIDH shared keys (or* 1 *in the case of leaves, as they have no children) using the neighbours' public keys:*

$$\begin{aligned} E_{\mathsf{par}(i),i} &= \phi_i^{\mathsf{par}(i)}(E_{\mathsf{par}(i)}) \\ &\cong E/\langle P_\iota + [r_i]Q_\iota, P_{1-\iota} + [r_{\mathsf{par}(i)}]Q_{1-\iota}\rangle, \\ E_{\mathfrak{l}.\mathsf{cld}(i),i} &= \phi_i^{\mathfrak{l}.\mathsf{cld}(i)}(E_{\mathfrak{l}.\mathsf{cld}(i)}) \\ &\cong E/\langle P_\iota + [r_i]Q_\iota, P_{1-\iota} + [r_{\mathfrak{l}.\mathsf{cld}(i)}]Q_{1-\iota}\rangle, \end{aligned}$$

for each $\mathfrak{l} \in \{1, \ldots, l_i\}$*, where*

$$\begin{aligned} \ker(\phi_i^{\mathsf{par}(i)}) &= \langle \phi_{\mathsf{par}(i)}(P_\iota) + [r_i]\phi_{\mathsf{par}(i)}(Q_\iota)\rangle, \\ \ker(\phi_i^{\mathfrak{l}.\mathsf{cld}(i)}) &= \langle \phi_{\mathfrak{l}.\mathsf{cld}(i)}(P_\iota) + [r_i]\phi_{\mathfrak{l}.\mathsf{cld}(i)}(Q_\iota)\rangle. \end{aligned}$$

publish$_2$: *Parties* $\mathcal{P}_0$ *and* $\mathcal{P}_1$ *have already computed the same session key* $K = K_0 = K_1$ *and send*

$$x_{\mathfrak{l}.\mathsf{cld}(0)} = K - j(E_{\mathfrak{l}.\mathsf{cld}(0),0}),$$

respectively,

$$x_{\mathfrak{l}.\mathsf{cld}(1)} = K - j(E_{\mathfrak{l}.\mathsf{cld}(1),1}),$$

to their respective children, for $\mathfrak{l} \in \{1, \ldots, l_0\}$*, respectively* $\mathfrak{l} \in \{1, \ldots, l_1\}$*.*

keygen AND publish$_3$: *Upon receiving* $x_{\mathsf{par}(i)}$*,* $\mathcal{P}_i$ *computes the session key*

$$K_i = x_{\mathsf{par}(i)} + j(E_{\mathsf{par}(i),i}).$$

Every party $\mathcal{P}_i$ *with children (this excepts the leaves of* Γ*), then computes* $x_{\mathfrak{l}.\mathsf{cld}(i)} = K_i - j(E_{\mathfrak{l}.\mathsf{cld}(i),i})$ *and multicasts this to its* $\mathfrak{l}$*-th child, for each* $\mathfrak{l} \in \{1, \ldots, l_i\}$*.*

It is easy to see that this protocol satisfies correctness: We have that $K_0 = K_1 = K$. Assume that $\mathcal{P}_{\mathsf{par}(i)}$ obtained the session key $K_{\mathsf{par}(i)} = K$. For party $\mathcal{P}_i$,

$$\begin{aligned} K_i &= x_{\mathsf{par}(i)} + j(E_{\mathsf{par}(i),i}) \\ &= (K - j(E_{i,\mathsf{par}(i)})) + j(E_{\mathsf{par}(i),i}) = K. \end{aligned}$$

The security of the protocol follows from an argument analogous to the proof given for Theorem 49.

Theorem 51. *Assuming the SSDDH problem is (post-quantum) hard, P2P-SIT is a (post-quantum) secure GKE, with forward-security.*

3.5 AUTHENTICATED SIT (A-SIT)

Consider simply signing each message sent in a key exchange. Intuitively, this should result in an authenticated key exchange as each message had been signed before being sent and can be cryptographically verified. If we extend this to GKEs, by combining it with a signature scheme, it should be possible to create an authenticated group key exchange (AGKE). Indeed, using a Katz-Yung compiler [41] (or even the improved version by Tang and Mitchell [61]), which has each party sign each message before sending, creates a constant-round AGKE.[5] Regrettably, those AGKEs resulting from such a compiling will have computational complexity $O(n)$ regardless of the underlying GKE (assuming linear or better complexity) as each party must perform verifications proportional to the total number of parties. To fix this handicap, we propose an adjusted version of the Katz-Yung compiler where the computational complexity remains equivalent to the underlying GKE. Our proposed compiler is a variant of the one given in Desmedt, Lange, and Burmester [18], which takes full advantage of the structure of the double-tree and the minimal party interactions.

As signature schemes usually require some form of public key infrastructures (PKIs) to link verification keys to parties, for example using a certificate authority for registration and issuance of certificates, we assume that such infrastructure is in place by the AGKE commencement.

In our adjusted Katz-Yung compiler, each party $\mathcal{P}_i$ must first generate static signing and verification keys. For each GKE instance, each involved party then generates session-specific randomness in the form of a random nonce. The party uses this nonce along with the party ID, session name, and message counter, for all communication.

In SIT, assuming a balanced double-tree, i.e. the same number of children per node, each user only needs to exchange a logarithmic number of messages in order to compute the session key. Our compiler preserves this communication complexity.

We assume there exists a set of all parties $\mathbb{P}$. We assume the existence of a secure signature scheme, Π_{sign}, and the forward-secure SIT, Π_{SIT}. We consider the corresponding parameter generating algorithms: Gen_{sign} and Gen_{SIT}. Gen_{sign} takes as input a security parameter $1^{\lambda_{sign}}$ and outputs parameters required for secure signing. Gen_{SIT} takes as input a security parameter $1^{\lambda_{SIT}}$ and a number of participants, n, and outputs the public parameters for SIT.

5 There exist other types of compilers for two-party authenticated key exchanges (see for example [37, 45]) that may possibly be extended into compilers for AGKEs as well, but such extensions are beyond the scope of the work included in this dissertation.

Protocol 52 (Authenticated SIT (A-SIT))**.** *The protocol is as follows.*

setup$_{\text{sign}}$: *For a security parameter $1^{\lambda_{sign}}$, the algorithm outputs to all of the parties in $\mathbb{P}$ the parameters for the chosen signature scheme:*

$$\mathfrak{P}_{Sign} \xleftarrow{R} \mathsf{Gen}_{sign}(1^{\lambda_{sign}}).$$

key$_{\text{sign}}$: *Given $\mathfrak{P}_{Sign}$, each party $\mathcal{P}_i \in \mathbb{P}$ generates the signing/verification keys $(\mathsf{sign}_i, \mathsf{vrfy}_i)$. These are static (long-term) keys.*

setup$_{\text{II}}$: *Let $\mathbb{P}_n = \{\mathcal{P}_0, \ldots, \mathcal{P}_{n-1}\} \subset \mathbb{P}$ be a set of n parties wishing to do a GKE and let gid be their group identifier (of size $O(\log_k n)$). Each $\mathcal{P}_i \in \mathbb{P}_n$ chooses an ephemeral random nonce $\eta_i \in \{0,1\}^{\lambda_{sign}}$ for the session.*

paragen: *For a security parameter $1^{\lambda_{SIT}}$ and the number of parties, n, the algorithm outputs to all the parties in $\mathbb{P}_n$ the parameters:*

$$(p, E, \{P_0, Q_0\}, \{P_1, Q_1\}, \Gamma, sN) \xleftarrow{R} \mathsf{Gen}_{SIT}(1^{\lambda_{SIT}}, n),$$

where sN is a unique session name.

where Γ includes the party arrangement, i.e. ancestors and descendants, and sN is the unique session name.

inisign: *Let $\mathsf{rel}_i^s = \{\mathcal{P}_{1'}, \mathcal{P}_{2'}, \ldots, \mathcal{P}_{t_i'}\}$ denote the set of all ancestors and the l_i children of $\mathcal{P}_i$ in the s-th session of party $\mathcal{P}_i$.[6] The size of this set depends, in particular, on the number of ancestors. Each $\mathcal{P}_i$ computes $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sign}_i}(\mathcal{P}_i|sN|0|\eta_i)$ and multicasts $\mathcal{P}_i|sN|0|\eta_i|\sigma$ to its parent and all its descendants.[7] After positive verification, each party $\mathcal{P}_i$ stores the session specific information as $\mathsf{info}_i^s = \mathsf{gid}|sN|\mathcal{P}_{1'}|\eta_{1'}|\ldots|\mathcal{P}_{t_i'}|\eta_{t_i'}$, as a part of the state information.*

SIT: *The SIT protocol, Π_{SIT}, is instantiated with these changes:*

- *Whenever party $\mathcal{P}_i$ is supposed to multicast a message m as part of the protocol, the party computes $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sign}_i}(\mathcal{P}_i|sN|j|m|\eta_i)$, where j is the message number, and multicasts[8] the concatenated string $\mathcal{P}_i|sN|j|m|\sigma$.*
- *Upon receiving $\mathcal{P}_*|sN|j|m|\sigma$, party $\mathcal{P}_i$ checks that:*
  1. *$\mathcal{P}_* \in \mathsf{rel}_i^s$,*
  2. *j is the next expected sequence number for messages from $\mathcal{P}_*$,*

6 This s may not be the same for each party in the protocol instance.

7 $\mathcal{P}_i|sN|0|\eta_i|\sigma$ can just be sent to all parties to eliminate the need to store a list of descendants, which in the case of $\mathcal{P}_0$ and $\mathcal{P}_1$ has size $O(n)$.

8 At most this requires one re-signing of a message when a party has to send the SIT protocol x value to a child (with which it has already exchanged more than one previous message) and its other descendants (with which it has sent at most one previous message).

3. $1 \leftarrow \mathsf{Vrfy}_{\mathsf{vrfy}_*}(\mathcal{P}_*|sN|j|m|\eta_*, \sigma)$.

If any of these are untrue, $\mathcal{P}_i$ aborts the protocol, ending the session, and wiping its state. Otherwise, $\mathcal{P}_i$ continues as it would in Π_{SIT} and uses m.

keygen: *Each non-aborted instance computes the session key as in Protocol 45.*

Theorem 53. *Assuming the signature scheme Π_{sign} is (post-quantum) secure and that SSDDH is (post-quantum) hard, A-SIT given in Protocol 52 is a (post-quantum) secure AGKE (with forward security).*

Proof. As a received message is utilized by a party as it would in Π_{SIT} if the message is verified, and so, by the correctness of SIT, the first requirement in Definition 42 is satisfied by Protocol 52.

For the second requirement, assume that there exists a (not necessarily classical) polynomial-time adversary $\mathcal{A}$ with non-negligible advantage $\mathsf{Adv}(\mathcal{A}) = \varepsilon$. We use the adversary $\mathcal{A}$ to construct a distinguisher $\mathcal{D}$ for SSDDH, which we give in Algorithm 2.

There are three ways $\mathcal{D}$ may succeed: By forging a signature for a party in the Test case, by reusing a signature, or by distinguishing the session key from random. The success probability of the first is bounded by the advantage of an adversary to forge a signature, i.e. negligible. The second is bounded by a combinatorial consideration of the number of sent messages and protocol executions compared to the space of random nonces. The third is bounded by the advantage of an adversary to distinguish between a SIT session key and a random j-invariant for a curve in the same space, which by Theorem 49 is negligible.

For any session, $\mathcal{D}$ has generated correctly distributed signing/verification keys and random nonces and so honestly simulates all communication in the protocol instances.

For any session $g \neq h$, $\mathcal{D}$ may answer any query by $\mathcal{A}$ entirely.

For the h-th session, the distributions of Algorithm 1 are used, which are distributed according to the SIT protocol as argued in the proof of Theorem 49, giving us the session key distributed as in the SSDDH problem. We therefore wish to show that the forging and repeating attacks succeed with negligible probability such that any distinguishing attack must occur on the key exchange part with non-negligible probability, i.e. we may build a SSDDH distinguisher.

Let Forge be the event that $\mathcal{A}$ can output a new, valid message/signature pair with respect to the public verification key vrfy_* of some party $\mathcal{P}_*$ *before* querying RevealStaticKey($\mathcal{P}_*$). The probability of this event occurring is bounded by the total number of parties times the success probability of forging the signature, $\mathsf{Succ}(\Pi_{sign})$, which per assumption was hard, i.e. negligible. In other words,

$$\Pr[\mathsf{Forge}] \leq |\mathbb{P}| \cdot Succ(\Pi_{sign}),$$

Algorithm 2 SSDDH distinguisher, $\mathcal{D}$.

Input: $E_0, E_1, \phi_0(P_1), \phi_0(Q_1), \phi_1(P_0), \phi_1(Q_0), E'$

1: $h \xleftarrow{R} \{1, \ldots, \Lambda\}$, where Λ is an upper bound on the number of sessions activated by $\mathcal{A}$ in any interaction.
2: Invoke $\mathcal{A}$ and simulate protocol to $\mathcal{A}$:
3: Generate static signature/verification keys for each party $\mathcal{P}_i \in \mathbb{P}$.
4: For the g-th session ($g \neq h$): Generate ephemeral random nonces as in $Setup_{II}$ and simulate the GKE protocol as prescribed by Protocol 52.
5: For the h-th session: Simulate as in the h-th session of Algorithm 1, using signature and verification keys and random nonces as prescribed by Protocol 52 for message sending.
6: **if** the h-th session is chosen by $\mathcal{A}$ as the test session **then**
7: Provide $\mathcal{A}$ as the answer to the distinguishing query, i.e.,
8: $d \leftarrow \mathcal{A}$'s output
9: **else**
10: $d \xleftarrow{R} \{0, 1\}$.

Output: d

such that the probability that Forge occurs, is negligible.

Let Repeat be the event that a nonce used by any party in response to a SEND query was previously used by that party. Recall that the adversary is in charge of all communications. If we ignore the uniqueness of the session name, sN, which will only force the SEND queries considered to be from the same execution of the protocol, then the probability of this event occurring is bounded by the maximal number of SEND queries in a protocol and the number of protocol executions, in the sense that

$$\Pr[\mathsf{Repeat}] \leq \frac{\nu(\nu + \Lambda)}{2^{\lambda_{sign}}},$$

where ν is the maximal number of SEND queries queried by the adversary in a single execution of the protocol, and Λ is as in Algorithm 2. As both ν and Λ are polynomial, this probability is negligible.

Let the sum of these two negligible probabilities, for Forge and Repeat, be denoted as negl.

If the test session is *not* the h-th session, then $\mathcal{D}$ outputs a random bit, i.e. has advantage 0. If the test session *is* the h-th session, which happens with probability $1/\Lambda$, then $\mathcal{A}$ will succeed with advantage ε. Hence, the final advantage of the SSDDH distinguisher $\mathcal{D}$ is $(\varepsilon - \mathsf{negl})/\Lambda$, which is non-negligible.

Due to the fact that our compiler simply adds signatures and signs every message, any adversary that could break forward security of A-SIT could break forward security of SIT, which has forward security,

by Theorem 49. This gives us a contradiction, hence A-SIT has forward security. □

The above compiler can be instantiated using P2P-SIT as the underlying protocol instead of SIT, with only a slight change to the Ini_{sign} step. The change is that party $\mathcal{P}_i$ still computes the signature $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sign}_i}(\mathcal{P}_i|sN|0|\eta_i)$ but only multicasts $\mathcal{P}_i|sN|0|\eta_i|\sigma$ to its *parent and its children*, as opposed to all the descendants. The session specific information is also reduced to

$$\mathsf{info}_i^s = \mathsf{gid}|sN|\mathcal{P}_{\mathsf{par}(i)}|\eta_{\mathsf{par}(i)}|\mathcal{P}_{1.\mathsf{cld}(i)}|\eta_{1.\mathsf{cld}(i)}|\ldots|\mathcal{P}_{l_i.\mathsf{cld}(i)}|\eta_{l_i.\mathsf{cld}(i)}.$$

By an analogous proof as that for A-SIT, we get the following theorem.

Theorem 54. *Assuming the signature scheme Π_{sign} is (post-quantum) secure and that SSDDH is (post-quantum) hard, authenticated P2P-SIT (A-P2P-SIT) is a (post-quantum) secure AGKE (with forward security).*

Note 55. *Initially, our GKE (and thereby the peer-to-peer version and AGKE versions) incorporates the generation of a double-tree and a session name, sN, into the protocol creation algorithm. The AGKE compiler of Sect. 3.5 is a variant of the Desmedt-Lange-Burmester compiler [18], where info_i^s (called direct_U^j in [18]) is no longer included in the signature of messages, but sN instead. The reason is that info_i^s has at least length $O(\log n)$ while the session name, sN, may be much shorter, which is an improvement without detracting from the security (see the Repeat event argument in the proof of Thm. 53). In fact, [18] do not use a session name as we do and do not sign their random nonces when multicasting them to the relevant parties, which leaves them open to attacks such as replay attacks.*

3.6 COMPARISON

In this section, we compare our GKE with a trivial isogeny based GKE, isogeny-based GKE (Isog-GKE) (sketched in Sect. 3.2), and SIBD. Essentially, in Isog-GKE, parties are assembled in a cyclic structure, as in BDI and SIBD, letting $\mathcal{P}_n = \mathcal{P}_0, \mathcal{P}_{n+1} = \mathcal{P}_1$, etc.. However, for Isog-GKE, each round each party computes an isogeny on the previous party's public curve using their own secret key. The party then uses the resulting curve as its public curve in the next round (see [27] for an explicit construction). The Isog-GKE and SIBD protocols both achieve n-party GKEs with $O(n)$ communication and memory complexity.

In the authenticated case, we compare an authenticated version of Isog-GKE given in [4], which we call isogeny-based AGKE (Isog-AGKE). We also compare with SIBD made into an AGKE by using the Katz-Yung compiler [41]. The Isog-AGKE only requires 3 rounds, however each round each party must compute many isogenies and send large amounts of information to the next party. Although it is slightly more efficient than a trivial isogeny based GKE with each message signed

Table 3.1: Detailed comparison table of isogeny-based GKEs, assuming a binary double-tree for SIT and P2P-SIT. **I** and **S** denote isogeny tuples and summed values, respectively. Values in square brackets are particular for leaf nodes, if they differ from other nodes.

Protocol	**Rounds**	**Isog.s**	**Public**	**Comm.**	**Values**
Trivial GKE	$n-1$	$n-1$	$(n-1)\mathbf{I}$	$n-1$	$n-1$
Isog-GKE [4]	2	$n-1$	$(n-1)\mathbf{I}$	$n-1$	$n-1$
SIBD [27]	2	3	$1\mathbf{I}+1\mathbf{S}$	$2\mathbf{I}+(n-1)\mathbf{S}$	$n-1$
SIT	2	4	$1\mathbf{I}+2\mathbf{S}$	$3\mathbf{I}+(\lfloor\log_2 n\rfloor-1)\mathbf{S}$	$\lfloor\log_2 n\rfloor$
		[2]	[1**I**]	[1**I** + $\lfloor\log_2 n\rfloor$**S**]	
P2P-SIT	$\lfloor\log_2 n\rfloor$	4	$1\mathbf{I}+2\mathbf{S}$	$3\mathbf{I}+1\mathbf{S}$	2
		[2]	[1**I**]	[1**I** + 1**S**]	

Table 3.2: Comparision overview of isogeny-based GKEs and AGKEs, assuming a balanced double-tree for SIT, P2P-SIT and their authenticated versions, with l children per non-leaf node.

Protocol	**Rounds**	**Isog.s**	**Public**	**Comm.**	**Values**
Trivial GKE	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Isog-GKE [4]	2	$O(n)$	$O(n)$	$O(n)$	$O(n)$
SIBD [27]	2	3	2	$O(n)$	$O(n)$
SIT	2	$l+2$	$l+1$	$O(\log_l n)$	$O(\log_l n)$
P2P-SIT	$O(\log_l n)$	$l+2$	$l+1$	$l+2$	2
Isog-AGKE [4]	3	$O(n)$	$O(n)$	$O(n)$	$O(n)$
SIBD [27] + Katz-Yung [41]	3	3	3	$O(n)$	$O(n)$
A-SIT	3	$l+2$	4	$O(\log_l n)$	$O(\log_l n)$
A-P2P-SIT	$O(\log_l n)$	$l+2$	$l+2$	$2l+3$	2

and verified, it is still an AGKE with communication and memory complexity $O(n)$.

We have already stated our comparison parameters before (number of rounds, communication complexity, and memory complexity) but for these isogeny based (A)GKEs, we will also compare the number of isogeny computations and the number of public values. We recall that the number of rounds is the maximal number of times any party must wait for information from other parties in order to proceed (this includes sequential rounds that a party may not be directly involved in). The communication complexity considers the maximal number of broadcast/multicast messages received by any party in one call of the protocol.[9] The memory complexity is the maximal number of memories needed to compute the session key. For our section specific values, the number of isogenies is the maximal number of isogenies computed per party and the number of public values is the maximal number of public values (keys, etc.) computed and broadcast/multicast per party (without multiplicity).

For SIT, P2P-SIT, and their authenticated versions, we note that having l_i children, party $\mathcal{P}_i$ must compute l_i j-invariant differences (the x-values generated in the protocol), one for each child, as well as $l_i + 2$ isogeny computations. As we do not a priori assume a balanced tree (where each non-leaf party has the same number of children), complexity analysis of our protocol is difficult beyond this. For comparison reasons, in this section, we therefore assume a balanced tree, i.e. $l_i = l$ for some $l \leq (n-2)/2$ for all non-leaf $\mathcal{P}_i$ ($\mathcal{P}_0$ and $\mathcal{P}_1$ have an extra child each, namely each other). As for l, computational and communicational complexity depends on l. However, the number of rounds depends on $\log l$. Thus, if computational complexity is critical, we use small l. If round complexity is critical, we use large l. In SIT, if we only consider computational power, then the fewest overall number of computations per party occurs when $l = 2$ ($l = 1$ would give linear order communication complexity). This is true of P2P-SIT protocol as well. On the other hand, in the authenticated versions, at most $l + \log_l n$ signatures must be verified per party and at most $l + 1$ messages signed. Considering this, larger l are preferable, computationally speaking. Table 3.1 shows a detailed comparison in parameters for isogeny-based GKEs, assuming fixed $l = 2$ for our proposed GKEs, while Table 3.2 shows a comparison for the GKEs and the AGKEs using big-O notation, assuming a balanced double-tree with l children per node for our protocols.

9 In doing so, it is assumed that broadcasting/multicasting a message is independent of the number of receivers but that receiving l messages means that the receiver incurs a cost of l, even if all messages are received in a single round. The reason for this is that it takes into account that receiving messages requires being online and also storing said messages while broadcasting/multicasting is usually a one-time operation.

Our SIT protocol consists of two rounds, where in the first round the SIDH public keys are exchanged and in the second round the differences of j-invariants are multicast. The number of isogenies per party is one initial one and then one for the parent and one for each child. The SIDH public key and one j-invariant difference per child (or no difference, in the case of the leaf-nodes) are public values multicast by each party while the SIDH public keys of the parent and each child, as well as one j-invariant difference from each ancestor, are the multicast values received. Finally, one j-invariant difference from each ancestor as well as the SIDH key shared with the parent are required by a party to compute the session key.

For P2P-SIT, the number of rounds are more or less exchanged with the communication and memory complexity.

For the authenticated versions, simply add a round for initializing the signature and increase the public values and communications for each of the extra public values needed to be multicast for the signature initialization (see Table 3.2).

For the computation of the shared session key, SIBD parties must receive and use one extra value from each of the n parties, while SIT parties only require one extra value per *ancestor*. This leads to SIT's low communication and memory complexity. On the other hand, in SIBD, each party only computes and broadcasts a single j-invariant product, while in SIT, each party computes and multicasts an extra j-invariant difference per child. For the sake of efficiency, especially when using an unbalanced double-tree in SIT, this must therefore be taken into account.

3.7 CONCLUDING REMARKS

We proposed supersingular isogeny tree-based GKE, a generalization of the tree-based BDII using SIDH, reducing the security to the SSDDH problem. We also proposed peer-to-peer SIT, a sequential version having the same security. Finally, we proposed authenticated SIT and authenticated P2P-SIT, AGKEs resulting from a modified Katz-Yung compiler that combined SIT, respectively P2P-SIT, with a signature scheme, retaining the low complexity of SIT, respectively P2P-SIT.

Versatility, low complexity, and security are the three things make SIT and P2P-SIT highly competitive GKEs. The versatility of tree-based constructions in the design of communications networks, where the double-tree may be structured according to processing power or memory capabilities of the parties, transfers to SIT and P2P-SIT. The trivial isogeny based GKE, Isog-GKE, and SIBD have no such versatility, forcing each party to shoulder equal work. SIT and P2P-SIT also allow for more or less equal work distribution (barring the leaves), but even in such a balanced version, where each non-leaf party has l children, SIT has $O(\log_l n)$ communication and memory complexity, as

opposed to $O(n)$, while P2P-SIT has constant communication and memory complexity. Lastly, the security proofs of SIT and P2P-SIT reduce simply and directly to the SSDDH problem, like SIDH, rather than a related problem, which the proof of SIBD does (see [60]), inspiring more confidence in the post-quantum security of SIT and P2P-SIT. The authenticated versions retain the versatility, low complexity, and security of these constructions as well. We are therefore confident in the utility of SIT, P2P-SIT, A-SIT, and A-P2P-SIT, in PQC.

On top of the utility of our own protocol, we hope that our construction helps to inspire trust in, and promote the use of, SIDH as a post-quantum secure KE.

4

GROUP KEY EXCHANGES FROM RING-LEARNING-WITH-ERRORS

4.1 INTRODUCTION

Ring-learning-with-errors (R-LWE) is a cryptographic approach for PQC, a candidate for maintaining security against the upcoming quantum computers while using classical computers to enact the cryptographic protocols. As is, it has been applied to fundamental cryptography such as the two party KEs by Ding, Xie, and Lin [20] (and with Peikert's reconciliation tweak [52]). Apon, Dachman-Soled, Gong, and Katz [3] at PQCrypto 2019, and Choi, Hong, and Kim [13] from 2020, both manage to extend the applicability further by showing how to create AGKEs from the D-R-LWE problem [48]. Their schemes are interesting generalizations of DH based GKEs, one by Burmester and Desmedt [8] and one by Dutta and Barua [23]. However, for n parties, their protocols both have linear order communication complexity.

In this chapter, we generalize BDII to the R-LWE framework (see Sect. 2.6) by combining it with the Ding et al [20] KE using Peikert's tweak [52], and call it the tree-based R-LWE GKE (Tree-R-LWE-GKE). Like SIT given in Chapter 3, it is a constant round protocol with logarithmic order communication and memory complexity. We also generalize the sequential version, the peer-to-peer tree-based R-LWE GKE (P2P-Tree-R-LWE-GKE). P2P-Tree-R-LWE-GKE achieves constant communication and memory complexity but logarithmic order round complexity. The security of these protocols reduces to a decision Diffie-Hellman-like version of the D-R-LWE problem, shown by Bos, Costello, Naehrig, and Stebila [7] to have comparable security to the D-R-LWE problem (see Theorem 38 on p. 29).

As we did in Chapter 3, we extend these protocols by creating a compiler that turns both of them into AGKEs, while maintaining logarithmic order complexities. The security of the compiler is given by a reduction to the decision Diffie-Hellman-like (DDH-like) problem (Defn. 37). The specific differences between our compiler and the one given in [18] are explained in our concluding remarks.

4.2 RELATED WORK

We give here a sketch of the Apon et al [3] and Choi et al. [13] R-LWE GKEs. The first is based on BDI directly while the second is based on a modification of BDI called Dutta-Barua. We give a sketch of the unauthenticated GKEs as given in [13], albeit in our notation.

Both GKEs arrange the parties in a ring structure, letting $\mathcal{P}_n = \mathcal{P}_0, \mathcal{P}_{n+1} = \mathcal{P}_1$, etc., and achieve post-quantum R-LWE n-party AGKEs with communication and memory complexity $O(n)$.

Protocol 56 ([3] and [13] R-LWE GKEs sketch)**.** *For n parties, the protocol is as follows.*

setup: *Upon being given a security parameter 1^λ, outputs a ring R_q and ring element $a \leftarrow R_q$ to all parties.*

publish$_1$: *Each party $\mathcal{P}_i$ chooses a 'small' secret value $s_i \in R_q$ and 'small' noise $e_i \in R_q$ and broadcasts $z_i = as_i + e_i$ to all other parties.*

publish$_2$: *Each party $\mathcal{P}_i$ chooses another 'small' noise $e'_i \in R_q$ and broadcasts $x_i = (z_{i+1} - z_{i-1})s_i + e'_i$ to all other parties.*

keygen: *Each protocol generates keys in the following ways, respectively.*

Apon+: $k_i = nz_{i-1}s_i + (n-1)x_i + (n-2)x_{i+1} + \cdots + x_{i+n-2}$.

Choi+: *Each party $\mathcal{P}_i$ calculates $y_i = x_i + z_{i-1}s_i$ and $y_{i+j} = x_{i+j} + y_{i+(j-1)}$ for $j = 1$ to $n-1$. Then $k_i = \sum_{j=0}^{n-1} y_{i+j}$.*

Needless to say, both GKEs have a linear complexity as keys and value need to be sent to and received from all the other parties in the instance. Both may be made into AGKEs using the Katz-Yung compiler [41].

4.3 R-LWE TREE-BASED GKE (TREE-R-LWE-GKE)

We again use the concept of a double-tree as given in Section 3.3 (p. 40). We recall that excepting the leaves of the tree, each party $\mathcal{P}_i$ has a parent $\mathsf{par}(i)$, and a set of children $\{\mathsf{l.cld}(i)|\mathfrak{l} = 1, 2, \ldots, l_i\}$ where $0 \leq l_i \leq n-2$ is the amount of children of $\mathcal{P}_i$, which are all considered the *neighbours* of $\mathcal{P}_i$ (see Figure 3.2). For all $\mathcal{P}_i$ that are leaves we have $l_i = 0$, i.e. the lowest rungs of each branch have no children. The set $\mathsf{ancestors}(i)$ is the set of indexes of all ancestors of a party $\mathcal{P}_i$, including i but having removed 0 and 1. Parties $\mathcal{P}_0$ and $\mathcal{P}_1$ are parents of each other.

For use in our AGKE compiler in Sect. 4.5, we add the *session name* to the GKE protocol parameter generation.

The tree-based R-LWE GKE (Tree-R-LWE-GKE) protocol for n parties, Π_n, takes as input the security parameter 1^λ and the number of parties, using the security parameter to determine the parameters m, R, q, R_q and χ for the DDH-like problem (Definition 37), and outputs a session key $K \in \{0,1\}^m$.

The parameter generator algorithm, $\mathsf{ParaGen}$, takes as input the security parameter 1^λ and the number of parties, n. The algorithm outputs a tuple consisting of the DDH-like parameters m, R, q, R_q, χ, a uniformly random $a \xleftarrow{R} R_q$, a double-tree for the n parties, Γ, and a unique session name, sN.

Protocol 57 (Tree-based R-LWE GKE (Tree-R-LWE-GKE))**.** *The parties $\mathcal{P}_i$ for $i = 0, 1, \ldots, n-1$ generate a GKE protocol Π_n as follows:*

setup: *Given a security parameter, 1^λ, and the number of parties, n, the algorithm outputs to each party $\mathcal{P}_i$ the tuple:*

$$\mathfrak{P} := (m, R, q, R_q, \chi, a, \Gamma, sN) \leftarrow \mathsf{ParaGen}(1^\lambda, n),$$

where sN is a unique session name.

publish$_1$: *Given $\mathfrak{P}$, each $\mathcal{P}_i$ individually and randomly chooses secret keys $s_i, e_i, e_i' \xleftarrow{R} \chi$ and then computes a corresponding public key $b_i = as_i + e_i$. $\mathcal{P}_i$ then multicasts its public key to its neighbours (parent and l_i children).*

publish$_{2a}$: *Upon receiving a public key $b_{\mathsf{par}(i)}$ from its parent, $\mathcal{P}_{\mathsf{par}(i)}$, $\mathcal{P}_i$ generates the value $v_i = b_{\mathsf{par}(i)}s_i + e_i'$. Using the randomized doubling function, $\mathcal{P}_i$ finds the value $\overline{v}_i \xleftarrow{R} \mathsf{dbl}(v_i) \in R_{2q}$. Then using the cross-rounding function, $\mathcal{P}_i$ computes*

$$c_i = \langle \overline{v}_i \rangle_{2q,2} \in \{0,1\}^m,$$

which is the reconciliation key for its parent. Finally, $\mathcal{P}_i$ sends this reconciliation key to said parent, $\mathcal{P}_{\mathsf{par}(i)}$.

We assume, wlog, that $\mathcal{P}_1$ generates c_1 and sends it to $\mathcal{P}_0$, while $\mathcal{P}_0$ generates no reconciliation key c_0.

publish$_{2b}$: *Upon receiving the respective reconciliation keys $c_{\mathsf{l.cld}(i)}$ from each of its l_i children, and using the value $\overline{v}_i$, $\mathcal{P}_i$ computes the respective shared keys $k_{\mathsf{par}(i),i}$ and $k_{\mathsf{l.cld}(i),i}$ for each $\mathfrak{l} \in \{1, \ldots, l_i\}$:*

$$k_{\mathsf{par}(i),i} = \lceil \overline{v}_i \rfloor_{2q,2} \in \{0,1\}^m,$$
$$k_{\mathsf{l.cld}(i),i} \leftarrow \mathsf{rec}(2b_{\mathsf{l.cld}(i)}s_i, c_{\mathsf{l.cld}(i)}) \in \{0,1\}^m,$$

using the modular rounding function, $\lceil \cdot \rfloor$, and the reconciliation function, rec.

Again, wlog, $\mathcal{P}_1$ sets $k_{0,1} = \lceil \overline{v_0} \rfloor_{2q,2} \in \{0,1\}^m$ while $\mathcal{P}_0$ computes $k_{1,0} \leftarrow \mathsf{rec}(2b_1s_0, c_1) \in \{0,1\}^m$.

publish$_3$: *Each $\mathcal{P}_i$ with children (this excepts the leaves of Γ) computes*

$$x_{\mathsf{l.cld}(i)} = k_{\mathsf{par}(i),i} \oplus k_{\mathsf{l.cld}(i),i},$$

and multicasts this value to its respective descendants, for each $\mathfrak{l} \in \{1, \ldots, l_i\}$.

keygen: *Each $\mathcal{P}_i$ computes the session key*

$$K_i = k_{\mathsf{par}(i),i} \oplus \bigoplus_{h \in \mathsf{ancestors}(i)} x_h = K.$$

Proposition 58 (Correctness)**.** *Except with negligible probability, each party in the Tree-R-LWE-GKE protocol (Protocol 57) computes the same session key* $K = k_{0,1}$.

Proof. Proof by induction. By key reconciliation, except with negligible probability, $K_0 = K_1 = k_{0,1}$. Assume that $K_{\mathsf{par}(i)} = K$ then, as

$$K_{\mathsf{par}(i)} = k_{\mathsf{par}(\mathsf{par}(i)),\mathsf{par}(i)} \oplus \bigoplus_{m \in \mathsf{ancestors}(\mathsf{par}(i))} x_m,$$

except with negligible probability, we have that, except with negligible probability,

$$\begin{aligned} K_i &= k_{\mathsf{par}(i),i} \oplus \bigoplus_{m \in \mathsf{ancestors}(i)} x_m \\ &= k_{\mathsf{par}(i),i} \oplus \left(k_{\mathsf{par}(\mathsf{par}(i)),\mathsf{par}(i)} \oplus k_{i,\mathsf{par}(i)} \right) \\ &\qquad \oplus \bigoplus_{m \in \mathsf{ancestors}(\mathsf{par}(i))} x_m \\ &= K_{\mathsf{par}(i)} = K. \end{aligned}$$

□

We draw attention to the session key being the shared key, $K_{0,1}$, of the initial parties $\mathcal{P}_0$ and $\mathcal{P}_1$.

Theorem 59. *Under the assumption that the D-R-LWE problem (Definition 30) is hard, the Tree-R-LWE-GKE protocol given in Protocol 57 is a secure GKE (with forward security).*

Proof. We must show that the protocol in Protocol 57 satisfies the security notion given in Definition 41. The first requirement is satisfied by the correctness shown in Proposition 58.

For the second requirement, assume that there exists a (not necessarily classical) polynomial-time adversary $\mathcal{A}$, allowed polynomially-many classical queries, with non-negligible advantage $\mathsf{Adv}(\mathcal{A}) = \varepsilon$ (see Definition 41 for this notation). We build a polynomial-time distinguisher $\mathcal{D}$, allowed polynomially-many classical queries, for the DDH-like problem in Algorithm 3.

As an analysis of our distinguishing algorithm, we note the following.

For every session, except the h-th, $\mathcal{D}$ simulates the Tree-R-LWE-GKE protocol to $\mathcal{A}$, choosing new random secret keys for each party in each session and simulating all communication through $\mathcal{A}$. As all randomness is generated anew for each session and there are no long-term keys, all sessions are independently generated. Hence, any attack on any other session does not reveal anything about the h-th session except through repetition of secret keys, which happens with negligible probability.

Algorithm 3 DDH-like distinguisher, $\mathcal{D}$.

Input: $(m, R, q, R_q, \chi, a, b_0, b_1, c, \kappa)$ as in the DDH-like problem.

1: $h \xleftarrow{R} \{1, \ldots, \Lambda\}$, where Λ is an upper bound on the number of sessions activated by $\mathcal{A}$ in any interaction.
2: Invoke $\mathcal{A}$ and simulate protocol to $\mathcal{A}$, except for the h-th activated protocol session.
3: For the h-th session:
4: Set $\mathfrak{P} := (m, R, q, R_q, \chi, a, \Gamma, sN)$, where Γ is an n-party double-tree and sN is the session name.
5: Set $b'_0 = b_0, b'_1 = b_1$ and $c_1 = c$. Choose $(s_i, e_i, e'_i) \xleftarrow{R} \chi^3$ for $i = 2, \ldots, n-1$ and set $b'_i = as_i + e_i$. Set $v_i = b_{\mathsf{par}(i)} s_i + e'_i$, generate $\overline{v}_i \xleftarrow{R} \mathsf{dbl}(v_i) \in R_{2q}$ and compute $c_i = \langle \overline{v}_i \rangle_{2q,2} \in \{0,1\}^m$. Simulate multicasting for each $\mathcal{P}_i$ along with identifying information $(\mathcal{P}_i, \mathfrak{s})$.
6: Set

$$x'_{\mathfrak{l}.\mathsf{cld}(0)} := \kappa \oplus k_{0,\mathfrak{l}.\mathsf{cld}(0)}, \quad \forall \mathfrak{l} \in \{1, 2, \ldots, l_0\},$$
$$x'_{\mathfrak{l}.\mathsf{cld}(1)} := \kappa \oplus k_{1,\mathfrak{l}.\mathsf{cld}(1)}, \quad \forall \mathfrak{l} \in \{1, 2, \ldots, l_1\},$$
$$x'_{\mathfrak{l}.\mathsf{cld}(i)} := k_{\mathsf{par}(i),i} \oplus k_{\mathfrak{l}.\mathsf{cld}(i),i}, \ \forall \mathfrak{l} \in \{1, 2, \ldots, l_i\},$$

for $i \geq 2$ where $\mathcal{P}_i$ is not a leaf in Γ.
7: **if** the h-th session is chosen by $\mathcal{A}$ as the test session **then**
8: Provide $\mathcal{A}$ as the answer to the test query,
9: $d \leftarrow \mathcal{A}$'s output
10: **else**
11: $d \xleftarrow{R} \{0,1\}$.

Output: d

For the h-th session, using the public information for $\mathcal{P}_0$, namely b_0, $\mathcal{D}$ simulates R-LWE KE (Definition 36) with the secret keys $s_{\mathfrak{l}.\mathsf{cld}(0)}, e_{\mathfrak{l}.\mathsf{cld}(0)}$, and $e'_{\mathfrak{l}.\mathsf{cld}(0)}$ of party $\mathcal{P}_{\mathfrak{l}.\mathsf{cld}(0)}$, obtaining the shared key $k_{0,\mathfrak{l}.\mathsf{cld}(0)} = k_{\mathfrak{l}.\mathsf{cld}(0),0}$, except with negligible probability. Likewise, using the public information for $\mathcal{P}_1$, namely b_1, $\mathcal{D}$ simulates R-LWE KE with the secret keys $s_{\mathfrak{l}.\mathsf{cld}(1)}, e_{\mathfrak{l}.\mathsf{cld}(1)}, e'_{\mathfrak{l}.\mathsf{cld}(1)}$ of party $\mathcal{P}_{\mathfrak{l}.\mathsf{cld}(1)}$, obtaining $k_{1,\mathfrak{l}.\mathsf{cld}(1)} = k_{\mathfrak{l}.\mathsf{cld}(1),1}$, except with negligible probability. All other shared keys may be computed in polynomial-time as the secret keys for $\mathcal{P}_i$ are known for $i = 2, \ldots, n-1$.

As the s_i, e_i, e'_i are chosen uniformly at random for $i \geq 2$, the distribution of the $b'_i, x'_{\mathfrak{l}.\mathsf{cld}(i)}$ in Algorithm 3 are identical to that in a Tree-R-LWE-GKE instance.

The transcript given to $\mathcal{A}$ by $\mathcal{D}$ is

$$(b'_0, \ldots, b'_{n-1}, x'_{1.\mathsf{cld}(0)}, x'_{2.\mathsf{cld}(0)}, \ldots, \ldots, x'_{l_0.\mathsf{cld}(0)}, x'_{1.\mathsf{cld}(1)}, \ldots, x'_{(l_{n-1}).\mathsf{cld}(n-1)}),$$

where we assign a blank value for the x' value when there is no child.

In each session, all attack queries are answered fully and honestly.

If the h-th session is the test session and κ is a valid Tree-R-LWE-GKE session key, then $\kappa = k_{0,1}$, i.e. (a, b_0, b_1, c, κ) is indeed a valid DDH-like tuple, where $\kappa = \lceil \overline{v}_0 \rfloor_{2q,2}$.

If the h-th session is *not* the test session, then $\mathcal{D}$ outputs a random bit, i.e. it has advantage 0. However, if the test session *is* the h-th session, which happens with probability $1/\Lambda$, then $\mathcal{A}$ will succeed with advantage ε. Hence, the final advantage of the DDH-like distinguisher $\mathcal{D}$ is ε/Λ, which is non-negligible. □

Corollary 60. *Assuming the D-R-LWE problem is post-quantum hard, the Tree-R-LWE-GKE protocol is a post-quantum secure GKE with forward security.*

4.4 PEER-TO-PEER R-LWE GROUP KEY EXCHANGE (P2P-TREE-R-LWE-GKE)

Here we give our peer-to-peer (sequential) version of Tree-R-LWE-GKE, calling it peer-to-peer tree-based R-LWE GKE (P2P-Tree-R-LWE-GKE). The protocol is sequential, which here means that a party completely recovers the session key before computing and sending any message to its children. As it is sequential and performed in such a manner, the number of rounds is bounded by the height of the double-tree but the communication and memory complexity become constant. Although overly similar to Tree-R-LWE-GKE, differences are found after step $Publish_{2b}$. Through an analogous proof as that given for Tree-R-LWE-GKE, the peer-to-peer version achieves post-quantum security, which we therefore omit.

Peer-to-peer tree-based R-LWE GKE (P2P-Tree-R-LWE-GKE) for n parties, Π_n, takes as input the security parameter 1^λ and the number of parties, using the security parameter to determine the parameters m, R, q, R_q and χ, as in the DDH-like problem (Definition 37), and outputs a session key $K \in \{0,1\}^m$.

The parameter generator algorithm, ParaGen, takes as input the security parameter 1^λ and the number of parties, n. The algorithm outputs a tuple consisting of the DDH-like parameters m, R, q, R_q, χ, a uniformly random $a \xleftarrow{R} R_q$, a double-tree for the n parties, Γ, and a unique session name, sN.

Protocol 61 (P2P-Tree-R-LWE-GKE)**.** *The parties $\mathcal{P}_i$ for $i = 0, 1, \ldots, n-1$ generate a GKE protocol Π_n as follows:*

setup: *Given the security parameter, 1^λ, and the number of parties, n, the algorithm outputs to each party $\mathcal{P}_i$ the tuple:*

$$\mathfrak{P} := (m, R, q, R_q, \chi, a, \Gamma, sN) \leftarrow \mathsf{ParaGen}(1^\lambda, n),$$

where sN is a unique session name.

publish$_{1}$: *Given $\mathfrak{P}$, each $\mathcal{P}_i$ individually and randomly chooses secret keys $s_i, e_i, e_i' \xleftarrow{R} \chi$ then computes a public key $b_i = as_i + e_i$. $\mathcal{P}_i$ then multicasts its public key to its neighbours (parent and l_i children).*

publish$_{2a}$: *Upon receiving a public key $b_{\mathsf{par}(i)}$ from its parent, $\mathcal{P}_{\mathsf{par}(i)}$, $\mathcal{P}_i$ generates the value $v_i = b_{\mathsf{par}(i)}s_i + e_i'$. Using the randomized doubling function, $\mathcal{P}_i$ finds the value $\overline{v}_i \xleftarrow{R} \mathsf{dbl}(v_i) \in R_{2q}$. Using the cross-rounding function, $\mathcal{P}_i$ then computes $c_i = \langle \overline{v}_i \rangle_{2q,2} \in \{0,1\}^m$, the reconciliation key for its parent, which $\mathcal{P}_i$ sends to said parent, $\mathcal{P}_{\mathsf{par}(i)}$.*

We assume, wlog, that $\mathcal{P}_1$ generates c_1 and sends it to $\mathcal{P}_0$, while $\mathcal{P}_0$ generates no reconciliation key c_0.

publish$_{2b}$: *Upon receiving the respective reconciliation keys $c_{\mathsf{l.cld}(i)}$ from its l_i children, and also using the value $\overline{v}_i$, $\mathcal{P}_i$ computes the shared keys $k_{\mathsf{par}(i),i}$ and $k_{\mathsf{l.cld}(i),i}$ for each $\mathfrak{l} \in \{1, \ldots, l_i\}$:*

$$k_{\mathsf{par}(i),i} = \lceil \overline{v}_i \rfloor_{2q,2} \in \{0,1\}^m,$$
$$k_{\mathsf{l.cld}(i),i} \leftarrow \mathsf{rec}(2b_{\mathsf{l.cld}(i)}s_i, c_{\mathsf{l.cld}(i)}) \in \{0,1\}^m,$$

for $\mathfrak{l} \in \{1, \ldots, l_i\}$, using the modular rounding function, $\lceil \cdot \rfloor$, and the reconciliation function, rec.

Again, wlog, $\mathcal{P}_1$ sets $k_{0,1} = \lceil \overline{v_0} \rfloor_{2q,2} \in \{0,1\}^m$ while $\mathcal{P}_0$ computes $k_{1,0} \leftarrow \mathsf{rec}(2b_1s_0, c_1) \in \{0,1\}^m$.

publish$_{3a}$: *Parties $\mathcal{P}_0$ and $\mathcal{P}_1$ have already computed the same session key $K = k_{1,0} = k_{0,1}$ (except with negligible probability) and send $x_{\mathsf{l.cld}(0)} = K \oplus k_{\mathsf{l.cld}(0),0}$, respectively $x_{\mathsf{l.cld}(1)} = K \oplus k_{\mathsf{l.cld}(1),1}$, to their respective children, for $\mathfrak{l} \in \{1, \ldots, l_0\}$, respectively $\mathfrak{l} \in \{1, \ldots, l_1\}$.*

keygen AND publish$_{3b}$: *Upon receiving $x_{\mathsf{par}(i)}$, $\mathcal{P}_i$ computes the session key*

$$K_i = x_{\mathsf{par}(i)} \oplus k_{\mathsf{par}(i),i}.$$

Every party $\mathcal{P}_i$ with children (this excepts the leaves of Γ), then computes $x_{\mathsf{l.cld}(i)} = K_i \oplus k_{\mathsf{l.cld}(i),i}$ and multicasts this to its $\mathfrak{l}$-th child, for each $\mathfrak{l} \in \{1, \ldots, l_i\}$.

Corollary 62. *Assuming the D-R-LWE problem is post-quantum hard, the P2P-Tree-R-LWE-GKE protocol (Protocol 61) is a post-quantum secure GKE with forward security.*

4.5 AUTHENTICATED TREE-R-LWE-GKE

In the compiler we give in this section, all parties $\mathcal{P}_i$ generate static (long-term) signing and verification keys before any GKE has been begun. For each GKE instance, each involved party then generates session-specific randomness in the form of a random nonce. The party

uses this nonce along with the party ID, session name, and message counter, for all communication.

As signature schemes usually require some form of public key infrastructures (PKIs) to link verification keys to parties, for example using a certificate authority for registration and issuance of certificates, we assume that such infrastructure is in place by the AGKE commencement.

In Tree-R-LWE-GKE, assuming a balanced double-tree, i.e. the same number of children per node, each user only needs to exchange a logarithmic number of messages in order to compute the session key. Our compiler preserves this communication complexity.

We assume there exists a set of all parties $\mathbb{P}$. Consider the following protocol, which uses a secure signature scheme, Π_{sign}, and the forward-secure Tree-R-LWE-GKE, Π_{tree}, where the latter is assumed to be instantiated on the parameters m, R, q, R_q, and χ as in the DDH-like problem (Problem 37). We consider the corresponding parameter generating algorithms: Gen_{sign} and Gen_{tree}. Gen_{sign} takes as input a security parameter $1^{\lambda_{sign}}$ and outputs parameters required for secure signing. Gen_{tree} takes as input 1^{λ} and a number of participants, n, and outputs the public parameters for Tree-R-LWE-GKE.

Protocol 63 (Authenticated Tree-R-LWE-GKE)**.** *The protocol is as following.*

setup$_{\mathbf{sign}}$: *For a security parameter $1^{\lambda_{sign}}$, the algorithm outputs to all of the parties in $\mathbb{P}$ the parameters for the chosen signature scheme:*

$$\mathfrak{P}_{sign} \xleftarrow{R} \mathsf{Gen}_{sign}(1^{\lambda_{sign}}).$$

key$_{\mathbf{sign}}$: *Given $\mathfrak{P}_{sign}$, each party $\mathcal{P}_i \in \mathbb{P}$ generates the signing/verification keys $(\mathsf{sign}_i, \mathsf{vrfy}_i)$. These are static (long-term) keys.*

setup$_{\mathbf{II}}$: *Let $\mathbb{P}_n = \{\mathcal{P}_0, \ldots, \mathcal{P}_{n-1}\} \subset \mathbb{P}$ be a set of n parties wishing to do a GKE (re-indexing if need be) and let gid be their group identifier (of size $O(\log_k n)$). Each $\mathcal{P}_i \in \mathbb{P}_n$ chooses an ephemeral random nonce $\eta_i \in \{0,1\}^{\lambda_{sign}}$ for the session.*

paragen: *For the security parameter 1^{λ} and the number of parties, n, the algorithm outputs to all the parties in $\mathbb{P}_n$ the Π_{tree} parameters:*

$$\mathfrak{P}_{tree} := (m, R, q, R_q, \chi, a, \Gamma, sN) \xleftarrow{R} \mathsf{Gen}_{tree}(1^{\lambda}, n),$$

the tuple of the public values for secure Tree-R-LWE-GKE, public value a, the double-tree Γ, and the unique session name sN, as prescribed by Tree-R-LWE-GKE.

ini$_{\mathbf{sign}}$: *Let $\mathsf{rel}_i^s = \{\mathcal{P}_{1'}, \mathcal{P}_{2'}, \ldots, \mathcal{P}_{t_i'}\}$ denote the set of all ancestors, party $\mathcal{P}_0$ or $\mathcal{P}_1$ (depending on which side of the double-tree $\mathcal{P}_i$ is on), and the l_i children of $\mathcal{P}_i$ in session s.[1] The size*

1 This s may not be the same for each party in the protocol instantiation.

of this set depends, in particular, on the amount of ancestors. Each $\mathcal{P}_i$ computes $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sign}_i}(\mathcal{P}_i|sN|0|\eta_i)$, and multicasts $\mathcal{P}_i|sN|0|\eta_i|\sigma$ to its parent and all its descendants. After positive verification, each party $\mathcal{P}_i$ stores the session specific information as $\mathsf{info}_i^s = \mathsf{gid}|sN|\mathcal{P}_{1'}|\eta_{1'}|\ldots|\mathcal{P}_{t'_i}|\eta_{t'_i}$, as a part of its state information.

GKE: *The Tree-R-LWE-GKE protocol, Π_{Tree}, is instantiated with these changes:*

- *Whenever party $\mathcal{P}_i$ is supposed to multicast a message m as part of the protocol, the party computes $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sign}_i}(\mathcal{P}_i|sN|j|m|\eta_i)$, where j is the message number, and multicasts the concatenated string $\mathcal{P}_i|sN|j|m|\sigma$.*[2]
- *Upon receiving $\mathcal{P}_*|sN|j|m|\sigma$, party $\mathcal{P}_i$ checks that:*
  1. *$\mathcal{P}_* \in \mathsf{rel}_i^s$,*
  2. *j is the next expected sequence number for messages from $\mathcal{P}_*$,*
  3. *$1 \leftarrow \mathsf{Vrfy}_{\mathsf{vrfy}_*}(\mathcal{P}_*|sN|j|m|\eta_*, \sigma)$.*

  If any of these are untrue, the session is aborted, i.e. $\mathcal{P}_i$ does not complete the session, wiping its state. Otherwise, $\mathcal{P}_i$ continues as it would in Π_{tree} and uses m.

keygen: *Each non-aborted session computes the session key as in Π_{tree}.*

Theorem 64. *Assuming that the signature scheme Π_{sign} is (post-quantum) secure, the D-R-LWE problem is (post-quantum) hard, the authenticated Tree-R-LWE-GKE given in Protocol 63 is a (post-quantum) secure AGKE (Definition 42), with forward security.*

Proof. As a message is utilized in a session as it would in Π_{tree} if the message is verified, by the correctness of Tree-R-LWE-GKE, and as we assume that Π_{sign} is secure (and thereby has correctness), the first requirement in Defn. 42 is satisfied by Prot. 63.

For the second requirement, assume that there exists a (not necessarily classical) polynomial-time adversary $\mathcal{A}$ with non-negligible advantage $\mathsf{Adv}(\mathcal{A}) = \varepsilon$. We use the adversary $\mathcal{A}$ to construct a distinguisher $\mathcal{D}$ for the DDH-like problem, which we give in Algorithm 4.

There are three ways $\mathcal{D}$ may succeed: By forging a signature for a party in the Test case, by reusing a signature, or by distinguishing a correct session key from random. The success probability of the first is bounded by the advantage of an adversary to forge a signature, i.e. negligible. The second is bounded by a combinatorial consideration of the amount of sent messages and protocol executions compared to the

2 At most this requires one re-signing of a message when a party is required to send the Tree-R-LWE-GKE protocol x value to a child (with which it has already exchanged more than one previous message) and its other descendants (with which it has sent at most one previous message).

space of random nonces. The third is bounded by the advantage of an adversary to distinguish between a Tree-R-LWE-GKE session key and a random key in the same space, which by Theorem 59 is negligible.

Algorithm 4 DDH-like distinguisher, $\mathcal{D}$.

Input: (a, b_0, b_1, c, κ)

1: $h \xleftarrow{R} \{1, \ldots, \Lambda\}$, where Λ is an upper bound on the number of sessions activated by $\mathcal{A}$ in any interaction.
2: Invoke $\mathcal{A}$ and simulate protocol to $\mathcal{A}$:
3: Generate static signature/verification keys for each party $\mathcal{P}_i \in \mathbb{P}$.
4: For every session except the h-th: Generate ephemeral random nonces and simulate GKE protocol Π_{tree} as prescribed by Prot. 63.
5: For the h-th session, using signature and verification keys and random nonces as prescribed by Prot. 63 for message sending/-multicasting:
6: Set $\mathfrak{P}_{tree} = (n, m, R, q, R_q, \chi, a, \Gamma, sN)$, where Γ is an n-party binary graph and sN is the session name.
7: Set $b'_0 = b_0, b'_1 = b_1$ and $c_1 = c$. Choose $(s_i, e_i, e'_i) \xleftarrow{R} \chi^3$ for $i = 2, \ldots, n-1$ and set $b'_i = as_i + e_i$. Set $v_i = b_{\mathsf{par}(i)} s_i + e'_i$, generate $\overline{v}_i \xleftarrow{R} \mathsf{dbl}(v_i) \in R_{2q}$ and compute $c_i = \langle \overline{v}_i \rangle_{2q,2} \in \{0,1\}^m$.
8: Set

$$x'_{j.\mathsf{cld}(0)} := \kappa \oplus k_{0,j.\mathsf{cld}(0)}, \quad \forall j \in \{1, 2, \ldots, l_0\},$$
$$x'_{j.\mathsf{cld}(1)} := \kappa \oplus k_{1,j.\mathsf{cld}(1)}, \quad \forall j \in \{1, 2, \ldots, l_1\},$$
$$x'_{j.\mathsf{cld}(i)} := k_{\mathsf{par}(i),i} \oplus k_{j.\mathsf{cld}(i),i}, \ \forall j \in \{1, 2, \ldots, l_i\},$$

for $i \geq 2$ where $\mathcal{P}_i$ is not a leaf in Γ.
9: **if** the h-th session is chosen by $\mathcal{A}$ as the test session **then**
10: Provide $\mathcal{A}$ as the answer to the distinguishing query,
11: $d \leftarrow \mathcal{A}$'s output
12: **else**
13: $d \xleftarrow{R} \{0,1\}$.

Output: d

DDH-like distinguisher $\mathcal{D}$ generates correctly distributed signing/verification keys and random nonces to be used in sessions, and using them, honestly simulates all communication requests in the sessions (recall that the adversary $\mathcal{A}$ is in charge of all communications in our model).

For any session $g \neq h$, $\mathcal{D}$ may answer any query by $\mathcal{A}$ entirely.

For the h-th session, the distributions are as in Tree-R-LWE-GKE, giving us the session key distributed as in the DDH-like problem. We therefore wish to show that the forging and repeating attacks succeed with negligible probability such that any distinguishing attack must occur

on the key exchange part with non-negligible probability, i.e. we may build a DDH-like distinguisher.

Let SignForge be the event that $\mathcal{A}$ can output a new, valid message/signature pair with respect to the public verification key vrfy_* of some party $\mathcal{P}_*$ *before* querying StaticReveal($\mathcal{P}_*$). The probability of this event occurring is bounded by the total number of parties times the success probability of forging the signature, $\mathsf{Succ}_{\mathcal{A}}(\Pi_{sign})$, which per assumption was hard, i.e. the probability is negligible. In other words,

$$\Pr[\mathsf{SignForge}] \leq |\mathbb{P}| \cdot Succ_{\mathcal{A}}(\Pi_{sign}),$$

such that the probability that SignForge occurs, is negligible.

Let Repeat be the event that a nonce used by any party in response to a Send query was previously used by that party. Recall that the adversary $\mathcal{A}$ is in charge of all communications. If we ignore the uniqueness of the session names, sN, which will only force the Send queries considered to be from the same execution of the protocol, then the probability of this event occurring is bounded by the maximum number of Send queries in a session and the amount of protocol executions, in the sense that

$$\Pr[\mathsf{Repeat}] \leq \frac{\nu(\nu+\Lambda)}{2^{\lambda_{sign}}},$$

where ν is the maximum number of Send queries queried by the adversary in a single execution of the protocol, and Λ is as in Algorithm 4. As both ν and Λ are polynomially-bounded, this probability is negligible.

Let the sum of these two negligible probabilities, for SignForge and Repeat, be denoted as negl.

In each session, all attack queries are answered fully and honestly.

If the test session is *not* the h-th session, then $\mathcal{D}$ outputs a random bit, i.e. has advantage 0. If the test session *is* the h-th session, which happens with probability $1/\Lambda$, then $\mathcal{A}$ will succeed with advantage ε. Hence, the final advantage of the DDH-like distinguisher $\mathcal{D}$ is $(\varepsilon - \mathsf{negl})/\Lambda$, which is non-negligible.

Forward security comes from the forward security of Π_{tree} combined with the ineffectiveness of replay attacks. □

The above compiler can be instantiated using the P2P-Tree-R-LWE-GKE as the underlying protocol instead of Tree-R-LWE-GKE, with only a slight change to the $\mathbf{ini_{sign}}$ step. The change is that party $\mathcal{P}_i$ still computes the signature $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sign}_i}(\mathcal{P}_i|sN|0|\eta_i)$ but only multicasts $\mathcal{P}_i|sN|0|\eta_i|\sigma$ to its *parent and its children*, as opposed to all the respective descendants. The session specific information is also reduced to

$$\mathsf{info}_i^s = \mathsf{gid}|sN|\mathcal{P}_{\mathsf{par}(i)}|\eta_{\mathsf{par}(i)}|\mathcal{P}_{1.\mathsf{cld}(i)}|\eta_{1.\mathsf{cld}(i)}|\cdots|\mathcal{P}_{l_i.\mathsf{cld}(i)}|\eta_{l_i.\mathsf{cld}(i)}.$$

By an analogous proof as that for authenticated Tree-R-LWE-GKE, we get the following theorem.

Table 4.1: Comparision overview of R-LWE based AGKEs.

Protocol	Rounds	Communication	Memory
Apon et al [3]	4	$O(n)$	$O(n)$
Choi et al [13]	3	$O(n)$	$O(n)$
Auth. Tree-R-LWE-GKE	4	$O(\log_2 n)$	$O(\log_2 n)$
Auth. P2P-tree-R-LWE-GKE	$O(\log_2 n)$	$O(\log_2 n)$	2

Theorem 65. *Assuming the signature scheme Π_{sign} is (post-quantum) secure and that the D-R-LWE problem is (post-quantum) hard, authenticated P2P-Tree-R-LWE-GKE is a (post-quantum) secure AGKE, with forward security.*

4.6 COMPARISON

In this section, we compare our AGKEs with other post-quantum R-LWE GKEs: Apon, Dachman-Soled, Gong, and Katz [3] and Choi, Hong, and Kim [13] (see Sect. 4.2).

We choose to consider our auth. Tree-R-LWE-GKE and auth. P2P-Tree-R-LWE-GKE having binary double-trees as their graphs, which are double-trees where each party (excepting leaves) has exactly 2 children. Due to the signature initialization, there is a communication overhead in both AGKE of at most $O(\log_2 n)$. This gives auth. Tree-R-LWE-GKE a constant number of rounds with communication and memory complexity $\log_2(n)$, while the values are more or less reversed for auth. P2P-Tree-R-LWE-GKE.

Let us reiterate our chosen evaluation parameters for the AGKEs (see Section 3.6, p. 52): the number of rounds, the communication complexity, and the number of values needed to compute the session key, i.e. the memory complexity. The number of *rounds* is the maximum number of times any party must wait for information from other parties in order to proceed. The *communication complexity* is the maximum number of broadcast/multicast messages received by any party in one call of the protocol. The *memory complexity* is the maximum number of values stored until the session key computation. Table 4.1 shows these parameters for our selected AGKEs.

Our auth. Tree-R-LWE-GKE consists of four rounds, where in the first round the signatures are initialized, in the second round the R-LWE public keys are exchanged, in the third round reconciliation keys are computed and exchanged, and in the fourth round the exclusive-or of shared keys are exchanged. The multicast values received during the protocol are the signature initialization values of all ancestors and each child, R-LWE public keys of the parent and each child, a reconciliation key from the parent, as well as one exclusive-or sum

from each ancestor. In order to compute the session key, each party must store one exclusive-OR sum from each ancestor as well as the R-LWE key shared with the parent until the final round.

The number of rounds, communication complexity, and values needed to generate the session key differ considerably between the auth. Tree-R-LWE-GKE and auth. P2P-Tree-R-LWE-GKE protocols. We remark that the overall smallest number of operations per party is achieved when the graph is constructed as a binary double-tree, in contrast to having multiple or variable children per node. However, if the structure of the network and the computational power of the parties is taken into consideration, among other factors, it might be advantageous to select one protocol over the other and also arrange the double-tree according to the factors.

Parameter Constraints. Beyond the parameter constraints required for the hardness of the D-R-LWE problem, the parameters of [3] and [13] (including the number of parties) are required to satisfy further bounds set by the key reconciliation and Rényi bounds, for correctness and security. Fixing the ring, noise distributions, and security parameters therefore limits the amount of parties their protocols can support, while our security proof sets no further constraints on our parameters and our correctness bound makes the amount of parties inconsequential (see below). Although our protocol does not have constraints other than those required for the hardness of the DDH-like problem, the advantage for an adversary in solving the DDH-like problem is less than the sum of the advantages of solving two instances of the D-R-LWE problem (see [7, Theorem 1]), meaning that our R-LWE parameters must be adjusted accordingly. For example, [7] suggest $n = 1024, q = 2^{32} - 1, \sigma = 8/\sqrt{2\pi}$ to achieve statistical 128-bit classical security, giving theoretical 64-bit post-quantum security, assuming Grover's algorithm corresponds to a square-root speed up to the search problem. Using these parameters and Proposition 2 of [7], we find that the failure rate of our two AGKEs are equivalent and bounded by the probability of at least one party having the wrong session key: $n \cdot 2^{-2^{14}}$.

4.7 CONCLUDING REMARKS

We gave a compiler for our Tree-R-LWE-GKE, relying on the hardness of the D-R-LWE problem (via the DDH-like problem) and the security of the signature scheme employed in the compiler. Our protocols give us versatile post-quantum R-LWE n-party AGKEs that, when balanced with 2 children per node, in one case achieves constant round complexity with communication and memory complexity $O(\log_2 n)$, and in the other case, constant memory complexity with round and communication complexity $O(\log_2 n)$.

We admit that although the R-LWE AGKEs of [3] and [13] have high communication and memory complexity, they may have some advantages as they integrate the R-LWE KE mechanics into the protocol steps. Our protocols require each pair of parent and child to complete a R-LWE KE before proceeding so it might be possible to improve our GKE by likewise integrating R-LWE KE mechanics into the tree structure. We have not yet considered the possibility. A great advantage of our protocols is that they are tree-based, thus benefiting the ability to structure the tree according to processing power and/or memory capabilities. We further consider that signatures might not be required for authentication if we employ an authenticated key exchange (AKE) as the underlying KE, such as the one proposed by Zhang et al [66], but the purpose of this chapter was to construct a compiler from the basic R-LWE KE.

In conclusion, the added security benefit from reducing to the indistinguishability of a single instance of Ding, Xie, and Lin's R-LWE KE with Peikert's tweak, the versatility of tree-based constructions, and most importantly, the low communication and memory complexity apparent in our constructions, make our protocols highly competitive R-LWE based PQC AGKEs.

5

GENERIC GROUP KEY EXCHANGE COMPILERS

5.1 INTRODUCTION

In secure communication, the cryptographic concept of a key exchange is indispensable. No secure key exchange equals no secure communication. This is why when Diffie and Hellman presented their famous Diffie-Hellman key exchange, they ushered in a new age of cryptography. Their key exchange relies on the difficulty of solving the DDH problem in order to create a public key exchange, the security of which itself relied on the hardness of the discrete logarithm problem (DLP). Since then, cryptographers have built on the foundation laid by this KE and have presented KEs similar in form, but based on other hard problems than DLP. Now, with the age of quantum computing slowly but surely approaching, the quest for a new world of secure cryptography has begun. On this quest, quantum secure cryptosystem alternatives to classical cryptosystems are the treasure at the end. However, due to the breadth and depth of modern cryptography, this is an gargantuan undertaking, and unfortunately, one with a deadline. Looking past the current needs, it also beckons the question, what shall we do when an even more powerful form of computing is emerges from the dark?

Collecting the efforts of the previous chapters, in this chapter, we generalize the work and present a GKE compiler that relies not on the security of an established KE currently available, but on the security of almost any underlying secure two-party KE to achieve its security. This is essentially what Desmedt, Lange, and Burmester set out to do in [18], however they only gave a proof for the DH based case and waved their hands at the more general compiler and security proof. We therefore give both a generic and generalized definition of a two-party KE as well as a related generic decisional hard problem before describing and proving the GKE compiler secure.

5.2 PRELIMINARIES

The following generic KE protocol definition is based on DH [19] (see Sect. 2.3.1), SIDH [25] (see Sect. 2.5), and R-LWE KE [20, 52] (see Sect. 2.6).

Public-key KEs are usually categorized as either being non-interactive or interactive. Both types begin with some handshaking: agreeing on a set of common and public parameters, which is usually assumed to be output to each party by a parameter generator, before any actual calculation begins. In a non-interactive protocol, parties then randomly choose a secret key and output a public key. They may then retrieve

the other party's public key at leisure and calculate the shared key. In an interactive protocol, after an initial public key is published, a round occurs wherein a party calculates a new public key using at least the initial public key of the other party. Yet another interaction round (by the party having the initial public key) may then occur, using this newest public key (if the newest public key is not used, we may consider the public key calculated to be part of the initial round, since all used information was already available). Such rounds may be repeated as necessary, until both parties are able to calculate a shared key.

As non-interactive KEs are not general enough for our purposes, and interactive protocols too general, we will need an inchoate concept of interaction. For this purpose, we define an encompassing concept, a 0/1-interactive KE (IKE). This definition encompasses non-interactive KEs (0-interactive KEs (0-IKEs)) and once-interactive key exchanges, 1-interactive KEs (1-IKEs) (or "one-sided" interactive key exchanges, as only one party is expected to publish a new public key, related to the other party's public key). Our definition can be generalized to arbitrary n, but as our main results only hold for $n = 0, 1$, we restrict our definition to these values. This also significantly simplifies our notation throughout.

In the below protocol, for the security parameter 1^λ, we consider the algorithm Gen that generates the public parameters for the key exchange. Furthermore, we consider the algorithms: secret key generators $\mathsf{Sec}, \mathsf{Sec}'$, public key generators $\mathsf{Pub}, \mathsf{Pub}'$, and Key_I and Key_A, the inactive party and active party session key generators, respectively.

Protocol 66 (0/1-interactive KE (0/1-IKE))**.** *The parties $\mathcal{P}_0$ and $\mathcal{P}_1$, generate a two-party key exchange protocol Π as follows:*

setup: *Given the security parameter, 1^λ, Gen outputs to both parties the tuple of public parameters:*

$$\mathfrak{P} = (\mathfrak{P}_0, \mathfrak{P}_1) \xleftarrow{R} \mathsf{Gen}(1^\lambda),$$

where $\mathfrak{P}_0$ and $\mathfrak{P}_1$ are party-specific tuples of public values.

publish: *Each party chooses a random secret key and uses it to compute a public key:*

$$sk_0 \xleftarrow{R} \mathsf{Sec}(\mathfrak{P}_0), \quad pk_0 \leftarrow \mathsf{Pub}(sk_0, \mathfrak{P}_0),$$
$$sk_1 \xleftarrow{R} \mathsf{Sec}(\mathfrak{P}_1), \quad pk_1 \leftarrow \mathsf{Pub}(sk_1, \mathfrak{P}_1),$$

where $sk_0, sk_1 \in \mathcal{K}_S$ and $pk_0, pk_1 \in \mathcal{K}_P$, i.e. elements of the primary secret key and public key spaces, respectively. Each party then sends their public key to the other party.

interact: *This round can be activated by either party. Designate the activating party the active party, $\mathcal{P}_A$, and the other the inactive party,*

$\mathcal{P}_I$ (for example, $\mathcal{P}_A := \mathcal{P}_1$ and $\mathcal{P}_I := \mathcal{P}_0$). Party $\mathcal{P}_A$, upon receiving pk_I, generates a second secret key and computes a second public key:

$$sk'_A \xleftarrow{R} \mathsf{Sec}'(sk_A, pk_I, \mathfrak{P}_A), \quad pk'_A \leftarrow \mathsf{Pub}'(sk_A, sk'_A, pk_I, \mathfrak{P}_A),$$

where $sk'_A \in \mathcal{K}_{S'}$ and $pk'_A \in \mathcal{K}_{P'}$, i.e. elements of the secondary secret key and public key spaces, respectively. $\mathcal{P}_A$ sends pk'_A to $\mathcal{P}_I$ and continues on to the next step.

keygen: *After receiving the necessary public keys, the session keys are calculated in the following ways.*

$$\mathcal{P}_I : k_I \leftarrow \mathsf{Key}_I(sk_I, pk_A, pk'_A, \mathfrak{P}_I),$$
$$\mathcal{P}_A : k_A \leftarrow \mathsf{Key}_A(sk_A, sk'_A, pk_I, \mathfrak{P}_A),$$

where k_I and k_A are elements of the session key space $\mathcal{K}$.

*If $k_I = k_A$, with non-negligible probability, then we say that the protocol satisfies **correctness**, i.e. $K = k_I = k_A$ is a shared key.*

Note 67. *Let us consider a few specificities of the above protocol definition.*

1. *In a non-interactive key exchange, the* **interact** *round is simply omitted, such that sk'_A and pk'_A are empty values, resulting in session keys:*

   $$\mathcal{P}_I : k_I \leftarrow \mathsf{Key}_I(sk_I, pk_A, \mathfrak{P}_I),$$
   $$\mathcal{P}_A : k_A \leftarrow \mathsf{Key}_A(sk_A, pk_I, \mathfrak{P}_A),$$

   as expected.

2. *Not all values need to be used in each of the algorithms but are presented in the definition for the sake of generality. It may also, for example, be the case that only one party publishes a public key in the* **publish** *round, while the other party acts as the active party in the* **interact** *round (leaving pk_A, and possibly sk_A, as empty values). It may also be the case that some values are equal, such as $\mathfrak{P}_0 = \mathfrak{P}_1$ or $K_S = K_{S'}$, depending on the specific key exchange, but have been indexed here for the sake of generality. The definition is left as general as possible to include as many key exchanges as possible, past and future.*

3. *The protocol definition says nothing of the security of the protocol and the only extraneous requirement is that it satisfy correctness.*

Taking the above definition and notation into use, we may define the generic decisional hard (GDH) problem that will be needed for our security reduction later. The definition is inspired to encompass the decisional problems used to prove the indistinguishability property of KEs and GKEs. It is primarily based on the (decisional) hardness problems for DH, SIDH, and R-LWE KE, i.e. the DDH (Defn. 15), SSDDH (Defn. 27), and DDH-like problem (Defn. 37).

The hardness problem can be defined for KEs and GKEs alike; the limiting factor is notational, not conceptual. For applicability in future protocols, where security reduces to either an underlying two-party KE or to a decisional problem on the GKE itself, we therefore define the generic decisional hard (GDH) problem to cover both types of hard problems.

Definition 68 (Generic decisional hard (GDH) problem)**.** *Consider a (G)KE protocol* Π *satisfying correctness. Given a tuple sampled with probability* $1/2$ *from one of the following two distributions:*

- $(\mathfrak{P}, \mathbf{pk}, \kappa)$*, where* $\mathfrak{P}$ *and* $\mathbf{pk}$ *are the party-specific public values and keys in* Π*, and* $\kappa \in \mathcal{K}$ *is the corresponding key in the session key space for* Π*,*
- $(\mathfrak{P}, \mathbf{pk}, \kappa)$*, where* $\mathfrak{P}$ *and* $\mathbf{pk}$ *are the party-specific public values and keys in* Π*, and* $\kappa \in \mathcal{K}'$ *is a key sampled uniformly at random from* $\mathcal{K}' \supseteq \mathcal{K}$*,*

determine from which distribution the tuple is sampled. If any probabilistic polynomial-time adversary solves this problem with at most negligible probability, we say that the GDH problem is ***hard****.*

Note 69. *The problem is considered in regards to the computational ability of the adversaries such that hardness w.r.t. classical/quantum/etc. adversaries gives corresponding classical/quantum/etc. hardness.*

Note 70. *For the* 0*-IKE, we generally have* $(\mathfrak{P}, \mathbf{pk}, \kappa) = (\mathfrak{P}, (pk_0, pk_1), \kappa)$*. For the* 1*-IKE:* $\mathfrak{P} = (\mathfrak{P}_0, \mathfrak{P}_1)$ *and* $\mathbf{pk} = (\mathbf{pk}_I, \mathbf{pk}_A) = (pk_0, (pk_1, pk'_A))$*.*

For an n-party GKE, we could potentially have

$$(\mathfrak{P}, \mathbf{pk}, \kappa) = ((\mathfrak{P}_0, \mathfrak{P}_1, \cdots, \mathfrak{P}_{n-1}), (\mathbf{pk}_0, \mathbf{pk}_1, \cdots, \mathbf{pk}_{n-1}), \kappa).$$

To ease the reader into these generic definitions and as an overview, we have compiled a table of KEs and their corresponding hard problems in the 0/1-IKE notation in Appendix A.

5.3 GENERALIZED GROUP KEY EXCHANGE COMPILER (GKE-C)

For our GKE compiler, it is absolutely crucial that the key space for the session keys, $\mathcal{K}$, is a subset of a group $(G, \cdot)$, i.e. a group with multiplicative operation,[1] so that we may manipulate session keys by using inverses and the group operation. It could also be a subset of a ring, but we require only the group superset property.

1 Note that "+" could be used instead, but the notation is pedagogical for our compiler. We also do not assume that the group is commutative. Furthermore, note that any set S can be made into a group, namely the free group, or universal group, generated by S, so this requirement is trivially satisfied. For computation purposes however, we assume that the group operation is efficient.

We would now like to define the GKE compiler. We yet again use the concept of a double-tree as given in Section 3.3 (p. 40), except that we now consider a binary version, as in BDII. We recall that excepting the leaves of the tree, each party $\mathcal{P}_i$ has a parent $\mathsf{par}(i)$, a left child $L.\mathsf{cld}(i)$, and a right child $R.\mathsf{cld}(i)$, which are all considered the *neighbours* of $\mathcal{P}_i$ (see Figure 3.2). For all $\mathcal{P}_i$ that are leaves we have no children. The set $\mathsf{ancestors}(i)$ is the set of indexes of all ancestors of a party $\mathcal{P}_i$, including i but having removed 0 and 1. Parties $\mathcal{P}_0$ and $\mathcal{P}_1$ are parents of each other.

We define **score** recursively as the following: $score(0) := 0$, $score(1) := 1$, and $score(i) = score(\mathsf{par}(i)) + 1$, where $i \geq 2$ is the index of the party $\mathcal{P}_i$. We assume that the score is calculated individually by the parties when the graph is fixed and does not occur as a bandwidth or computational cost in our protocol. Using this score, we define the map $\iota = \iota(i) := score(i) \pmod 2$. Score itself will not be used explicitly, but ι is used in our security proof for Theorem 74 below.

The GKE compiler (GKE-C) for n parties, Π_n, given below, takes as input a security parameter 1^λ and a 0/1-IKE protocol, Π, (using the same security parameter) including the algorithms: $\mathsf{Gen}, \mathsf{Sec}, \mathsf{Sec}', \mathsf{Pub}, \mathsf{Pub}'$, and $\mathsf{Key}_I, \mathsf{Key}_A$, the public parameters generator, the secret key generators, the public key generators, and the session key generators. GKE-C outputs a shared key $k \in \mathcal{K}$ where $\mathcal{K} \subseteq G$ is the same key space as the session key space of Π and is a subset of a group $(G, \cdot)$.

The multi-party parameter generator algorithm, $\mathsf{Gen}_{\mathsf{mp}}$, is used to decide public parameters for the GKE protocol. It takes as input the security parameter, 1^λ, the number of parties, n, and an 0/1-IKE protocol, Π, and outputs a tuple consisting of public parameters for each party $\mathcal{P}_i$ and a double-tree for the n parties.

Protocol 71 (GKE compiler)**.** *Parties $\mathcal{P}_i$, for $i = 0, 1, \ldots, n-1$, generate a GKE protocol Π_n as follows:*

setup: *For the security parameter 1^λ, number of parties n, and two-party 0/1-IKE protocol Π, the algorithm outputs to each party $\mathcal{P}_i$ the tuple:*

$$\mathfrak{P} = ((\mathfrak{P}_0, \mathfrak{P}_1), \Gamma) \leftarrow \mathsf{Gen}_{\mathsf{mp}}(1^\lambda, n, \Pi),$$

where $(\mathfrak{P}_0, \mathfrak{P}_1)$ is as given by $\mathsf{Gen}(1^\lambda)$ in Π and Γ is a double-tree.

publish$_1$: *Each $\mathcal{P}_i$ chooses a random secret key $sk_i \xleftarrow{R} \mathsf{Sec}(\mathfrak{P}_\iota)$ and generates a public key $pk_i \leftarrow \mathsf{Pub}(sk_i, \mathfrak{P}_\iota)$. $\mathcal{P}_i$ then multicasts its public key to its neighbours (parent and up to two children).*

If using a 1-IKE, each child acts as the active party with its parent acting as the inactive party.[2] Regardless, the parties continue and complete their respective key exchanges, culminating in the shared keys for each i: $k_{\mathsf{par}(i),i}$, $k_{L.\mathsf{cld}(i),i}$, and $k_{R.\mathsf{cld}(i),i}$, between $\mathcal{P}_i$ and the parent, left child, and right child, respectively.

2 This means that at most a single secondary key is chosen per party as each party has only a single parent.

publish$_2$: *Each $\mathcal{P}_i$ with children computes $x_{L.\mathsf{cld}(i)} = (k_{L.\mathsf{cld}(i),i})^{-1} \cdot k_{\mathsf{par}(i),i}$ and $x_{R.\mathsf{cld}(i)} = (k_{R.\mathsf{cld}(i),i})^{-1} \cdot k_{\mathsf{par}(i),i}$, and multicasts these to all its left and right descendants, respectively.*

keygen: *Each $\mathcal{P}_i$ computes a shared key*

$$K_i = k_{\mathsf{par}(i),i} \cdot \prod_{j \in \mathsf{ancestors}(i)} x_j = k_{0,1} = K.$$

Proposition 72 (Correctness)**.** *Each party in GKE-C (Prot. 71) computes the same key $K = k_{0,1}$, with non-negligible probability.*

Proof. This can be seen by induction. Obviously, $K = K_0 = K_1 = k_{0,1}$ with non-negligible probability. Assume that $K_{\mathsf{par}(i)} = K$, then, as

$$K_{\mathsf{par}(i)} = k_{\mathsf{par}(\mathsf{par}(i)),\mathsf{par}(i)} \cdot \prod_{j \in \mathsf{ancestors}(\mathsf{par}(i))} x_j,$$

we have that

$$\begin{aligned} K_i &= k_{\mathsf{par}(i),i} \cdot \prod_{j \in \mathsf{ancestors}(i)} x_j \\ &= k_{\mathsf{par}(i),i} \cdot (k_{i,\mathsf{par}(i)})^{-1} \cdot k_{\mathsf{par}(\mathsf{par}(i)),\mathsf{par}(i)} \cdot \prod_{j \in \mathsf{ancestors}(\mathsf{par}(i))} x_j \\ &= K_{\mathsf{par}(i)} = K, \end{aligned}$$

with non-negligible probability. □

The session key of the group is equal to the shared key of the initial parties $\mathcal{P}_0$ and $\mathcal{P}_1$, so the security of the GKE must rely on the security of the 0/1-IKE in some form.

Note 73. *Although we can define the GKE-C for variable children per node, just like the constructions in the previous chapters, for ease of proof and explanation we consider all the nodes to have the same number of children, excepting the leaves and initial two parties, namely a balanced double-tree with two children per node.*

For our security proof of the GKE-C, we show a reduction to the GDH problem. It is a simple reduction showing that breaking the GDH problem for the GKE-C equates to breaking the GDH problem for the underlying two-party 0/1-IKE. This is the same as reducing the (in)distinguishability notion of our security model given in Sect. 2.8.1 to the GDH problem for the underlying 0/1-IKE. For this reason, be mindful that our GKE-C can at best have the same security as the underlying 0/1-IKE.

Theorem 74. *Suppose we have an 0/1-IKE, Π, with security parameter 1^λ and algorithms $\mathsf{Gen}, \mathsf{Sec}, \mathsf{Sec}', \mathsf{Pub}, \mathsf{Pub}'$, and $\mathsf{Key}_I, \mathsf{Key}_A$. If the 0/1-IKE satisfies correctness and reduces to an instance of the GDH problem that is hard, then GKE-C, with security parameter 1^λ and using Π as its underlying 0/1-IKE, is a secure GKE.*

Proof. We must show that Prot. 71 satisfies the security notion given in Defn. 41. For the proof, we consider the more complicated case of using a 1-IKE. The first requirement is satisfied by the correctness shown in Prop. 72.

For the second requirement, assume that there exists a (not necessarily classical) polynomial-time adversary $\mathcal{A}$ with non-negligible advantage $\mathsf{Adv}(\mathcal{A}) = \varepsilon$, distinguishing between the correct session key in a GKE-C instance and a random key in the key space, which is by definition the same as that of the underlying 1-IKE. We build a polynomial-time distinguisher $\mathcal{D}$ for the GDH problem (Definition 68) for an instance of 1-IKE, i.e. the security of GKE-C reduces to the security of 1-IKE. The distinguisher is given in Algorithm 5.

As an analysis of our distinguishing algorithm, we note the following. For the h-th session, using the public value $\mathfrak{P}_0$ for $\mathcal{P}_0$ and $\mathfrak{P}_{\iota(2)} = \mathfrak{P}_1$ for $\mathcal{P}_2$, the algorithm completes a 1-IKE instance with the secret key $\mathbf{sk}_2 = (sk_2, sk'_2)$ of party $\mathcal{P}_2$, giving the shared key $k_{0,2} = k_{2,0}$, with non-negligible probability, as we have assumed that the 1-IKE satisfies correctness. Likewise, it finds $k_{0,3} = k_{3,0}$, using $\mathbf{sk}_3 = (sk_3, sk'_3)$ and the public values $\mathfrak{P}_{\iota(3)}$ of $\mathcal{P}_3$. Using the public values $\mathfrak{P}_1$ for $\mathcal{P}_1$, and public values $\mathfrak{P}_{\iota(4)}$, it completes a 1-IKE instance with the secret key $\mathbf{sk}_4 = (sk_4, sk'_4)$ of party $\mathcal{P}_4$, giving the shared key $k_{1,4} = k_{4,1}$, and likewise $k_{1,5} = k_{5,1}$ using $\mathbf{sk}_5 = (sk_5, sk'_5)$ and the public values $\mathfrak{P}_{\iota(5)}$. All other shared keys may be computed as the secret keys for $\mathcal{P}_i$ are known for $i = 2, \ldots, n-1$.

As the $\mathbf{sk}_i$ are chosen uniformly at random for $i \geq 2$, the distribution of the $\mathbf{pk}_i = (pk_i, pk'_i)$ and $\mathbf{x}'_i = (x'_{L.\mathsf{cld}(i)}, x'_{R.\mathsf{cld}(i)})$ are identical to that in a GKE-C instance.

The transcript given to $\mathcal{A}$ by $\mathcal{D}$ is $(\mathbf{pk}_0, \ldots, \mathbf{pk}_{n-1}, \mathbf{x}'_0, \mathbf{x}'_1, \ldots, \mathbf{x}'_{n-1})$, where we assign a blank value for the $\mathbf{x}'$ value when there is no child.

If the h-th session is $\mathcal{A}$'s test session, then $\mathcal{D}$ is issuing $K = \kappa$, as $\mathcal{A}$'s test key. If K is a valid session key for the GKE-C, then $\kappa = K = k_{0,1} = k_{1,0}$, i.e. $(\mathfrak{P} = (\mathfrak{P}_0, \mathfrak{P}_1), \mathbf{pk} = (pk_0, (pk_1, pk'_A)), \kappa)$ is indeed a valid GDH tuple for the underlying two-party 1-IKE protocol Π and can be distinguished.

If the test session is *not* the h-th session, then $\mathcal{D}$ outputs a random bit, i.e. it has advantage 0. If the test session *is* the h-th session, which happens with probability $1/\eta$, then $\mathcal{A}$ will succeed with advantage ε. Hence, the final advantage of the GDH distinguisher $\mathcal{D}$ is ε/η, which is non-negligible. □

5.4 PEER-TO-PEER GKE-C (P2P-GKE-C)

We here also give a peer-to-peer version of our compiler as in the preceding chapters, calling it the the peer-to-peer GKE compiler (P2P-GKE-C), and review the differences between it and the previously given GKE compiler. A benefit of our peer-to-peer version is that it is

Algorithm 5 GDH distinguisher, $\mathcal{D}$.

Input: $\mathfrak{P} = (\mathfrak{P}_0, \mathfrak{P}_1)$, $\mathbf{pk} = (\mathbf{pk}_I, \mathbf{pk}_A) = (pk_0, (pk_1, pk'_A))$, and κ

1: $h \leftarrow \{1, \ldots, \Lambda\}$ uniformly chosen, where Λ is an upper bound on the number of sessions activated by $\mathcal{A}$ in any interaction.
2: Invoke $\mathcal{A}$ and simulate protocol to $\mathcal{A}$, except for the h-th activated protocol session.
3: For the h-th session:
4: Set the public parameters as $(\mathfrak{P} = (\mathfrak{P}_0, \mathfrak{P}_1), \Gamma)$, where Γ is an n-party double-tree.
5: Using $\mathcal{A}$ to relay messages (such as public values), simulate the 0/1-IKE between $\mathcal{P}_0$ and $\mathcal{P}_1$ using $(\mathfrak{P}_0, \mathfrak{P}_1)$, i.e. by setting the inactive party to be $\mathcal{P}_0$ and the active party to be $\mathcal{P}_1$.

Generate $sk_i \xleftarrow{R} \mathsf{Sec}(\mathfrak{P}_\iota)$ uniformly at random for $i = 2, \ldots, n-1$, and set $pk_i \leftarrow \mathsf{Pub}(sk_i, \mathfrak{P}_\iota)$. Simulate multicasting using $\mathcal{A}$ and follow protocol Π for each $\mathcal{P}_i$ and neighbour, letting the parent in each pair act as the inactive party and the children act as the active parties, generating secondary secret and public keys as needed.

Eventually, for each neighbour of $\mathcal{P}_i$, $i = 2, \cdots, n-1$, there exists shared keys available to $\mathcal{P}_i$: $k_{\mathsf{par}(i),i}$, $k_{L.\mathsf{cld}(i),i}$, and $k_{R.\mathsf{cld}(i),i}$.
6: Set

$$\begin{aligned} x'_{L.\mathsf{cld}(0)} &:= (k_{0,2})^{-1} \cdot \kappa, & x'_{R.\mathsf{cld}(0)} &:= (k_{0,3})^{-1} \cdot \kappa, \\ x'_{L.\mathsf{child}(1)} &:= (k_{1,4})^{-1} \cdot \kappa, & x'_{R.\mathsf{cld}(1)} &:= (k_{1,5})^{-1} \cdot \kappa, \text{ and} \\ x'_{L.\mathsf{cld}(i)} &:= (k_{L.\mathsf{cld}(i),i})^{-1} \cdot k_{\mathsf{par}(i),i}, \\ x'_{R.\mathsf{cld}(i)} &:= (k_{R.\mathsf{cld}(i),i})^{-1} \cdot k_{\mathsf{par}(i),i}, \end{aligned}$$

for $i \geq 2$ where $\mathcal{P}_i$ is not a leaf in Γ. Simulate multicasting for each applicable $\mathcal{P}_i$.
7: **if** the h-th session is chosen by $\mathcal{A}$ as the test session **then**
8: Provide $\mathcal{A}$ as the answer to the test query,
9: $d \leftarrow \mathcal{A}$'s output
10: **else**
11: $d \xleftarrow{R} \{0, 1\}$ uniformly at random.

Output: d

sequential and can therefore be used when memory and communication complexity need to be reduced due to, for example, bandwidth restrictions. We note that the number of rounds and the communication complexity switches, the communication complexity becoming constant and the number of rounds becoming logarithmic. For our two protocols, differences between them begin after the $\mathbf{publish_1}$ round.

The protocol P2P-GKE-C for n parties, Π_n^{P2P}, takes as input a security parameter 1^λ and an 0/1-IKE protocol, Π, (using the same

security parameter) including the algorithms: $\mathsf{Gen}, \mathsf{Sec}, \mathsf{Sec}', \mathsf{Pub}, \mathsf{Pub}'$, and $\mathsf{Key}_I, \mathsf{Key}_A$, the public parameters generator, the secret key generators, the public key generators, and the session key generators. The P2P-GKE-C outputs a shared key $k \in \mathcal{K}$ where $\mathcal{K} \subseteq \mathcal{G}$ is the same key space as the session key space of Π and is a subset of a group $(\mathcal{G}, \cdot)$.

The multi-party parameter generator algorithm, $\mathsf{Gen}_{\mathsf{mp}}$, is used to decide public parameters for the GKE. It takes as input the security parameter, 1^λ, the number of parties, n, and an $0/1$-IKE protocol, Π, and outputs a tuple consisting of public parameters for each party $\mathcal{P}_i$ and a double-tree, Γ, for the n parties.

Protocol 75 (Peer-to-peer GKE compiler)**.** *The parties $\mathcal{P}_i$, for $i = 0, 1, \ldots, n-1$, generate a GKE Π_n^{P2P} as follows:*

setup: *For the security parameter 1^λ, number of parties n, and $0/1$-IKE Π, the algorithm outputs to each party $\mathcal{P}_i$ the tuple:*

$$\mathfrak{P} = ((\mathfrak{P}_0, \mathfrak{P}_1), \Gamma) \leftarrow \mathsf{Gen}_{\mathsf{mp}}(1^\lambda, n, \Pi),$$

where $(\mathfrak{P}_0, \mathfrak{P}_1)$ is as given by $\mathsf{Gen}(1^\lambda)$ in Π and Γ is a binary double-tree.

publish$_1$: *Each $\mathcal{P}_i$ chooses a random secret key $sk_i \xleftarrow{R} \mathsf{Sec}(\mathfrak{P}_\iota)$ and generates a public key $pk_i \leftarrow \mathsf{Pub}(sk_i, \mathfrak{P}_\iota)$. $\mathcal{P}_i$ then multicasts its public key to its neighbours (parent and up to two children).*

If using a 1-IKE, each child acts as the active party with its parent acting as the inactive party.[3] Regardless, the parties continue and complete their respective KEs, culminating in the shared keys for each i: $k_{\mathsf{par}(i),i}$, $k_{L.\mathsf{cld}(i),i}$, and $k_{R.\mathsf{cld}(i),i}$, between $\mathcal{P}_i$ and the parent, left child, and right child, respectively.

publish$_2$: *Parties $\mathcal{P}_0$ and $\mathcal{P}_1$ have already computed the same session key $K = k_{0,1} = K_0 = K_1$ (with non-negligible probability) and send*

$$\begin{aligned} x_{L.\mathsf{cld}(0)} &= K \cdot (k_{L.\mathsf{cld}(0),0})^{-1}, \text{ respectively} \\ x_{L.\mathsf{cld}(1)} &= K \cdot (k_{L.\mathsf{cld}(1),1})^{-1}, \text{ and} \\ x_{R.\mathsf{cld}(0)} &= K \cdot (k_{R.\mathsf{cld}(0),0})^{-1}, \text{ respectively} \\ x_{R.\mathsf{cld}(1)} &= K \cdot (k_{R.\mathsf{cld}(1),1})^{-1}, \end{aligned}$$

to their left, respectively right, children.[4]

keygen AND **publish$_3$:** *Upon receiving $x_{\mathsf{par}(i)}$, $\mathcal{P}_i$ computes the session key*

$$K_i = x_{\mathsf{par}(i)} \cdot k_{\mathsf{par}(i),i}.$$

3 This means that at most a single secondary key is chosen per party as each party has only a single parent.

4 The products in these x values could also be reversed, as long as the rest of the procedure remains consistent, for example in the **keygen** and **publish$_3$** round, regardless of the commutativity of the group.

Table 5.1: Examples of GKE-C compiled GKEs.

	BDII [18]	**SIT** [32]	**Tree-R-LWE-GKE** [33]
0/1-**IKE**	DH [19]	SIDH [25]	R-LWE w/ tweak [20, 52]
GDH	DDH [19]	SSDDH [25]	DDH-like [7]

> *Each party $\mathcal{P}_i$ with children (this excepts the leaves of Γ), then computes $x_{L.\mathsf{cld}(i)} = K_i \cdot (k_{L.\mathsf{cld}(i),i})^{-1}$ and $x_{R.\mathsf{cld}(i)} = K_i \cdot (k_{R.\mathsf{cld}(i),i})^{-1}$ and multicasts this to its left, respectively right, child.*

It is easy to see that this protocol satisfies correctness: We have that $K_0 = K_1 = K$ with non-negligible probability. Assume that $\mathcal{P}_{\mathsf{par}(i)}$ obtained the session key $K_{\mathsf{par}(i)} = K$. For party $\mathcal{P}_i$, we have $K_i = x_{\mathsf{par}(i)} \cdot k_{\mathsf{par}(i),i} = K_{\mathsf{par}(i)} \cdot (k_{i,\mathsf{par}(i)})^{-1} \cdot k_{\mathsf{par}(i),i} = K$, with non-negligible probability. Security follows from an analogous argument to that of the proof for Thm. 74.

Theorem 76. *Suppose we have an* 0/1*-IKE protocol,* Π*, with security parameter* 1^λ *and algorithms* $\mathsf{Gen}, \mathsf{Sec}, \mathsf{Sec}', \mathsf{Pub}, \mathsf{Pub}'$*, and* $\mathsf{Key}_I, \mathsf{Key}_A$*. If the* 0/1*-IKE satisfies correctness and reduces to an instance of the GDH problem, then P2P-GKE-C, with security parameter* 1^λ *and using* Π *as its underlying* 0/1*-IKE, is a secure GKE.*

Parties in P2P-GKE-C have the same session key as in GKE-C, namely the shared key of parties $\mathcal{P}_0$ and $\mathcal{P}_1$.

As our compilers require a 0/1-IKE and GDH problem, for the 0/1-interactive KEs (IKEs) explained in this dissertation and their associated hard problems, Table 5.1 gives an overview of the (literature of the) GKEs that result from them.

5.5 COMPLEXITY ANALYSIS

In this section, we consider the computation and communication complexities of our GKE-Cs.

We consider the following comparison parameters, similar to those in the previous chapters (see Section 3.6, p. 52): round complexity, number of public values, communication complexity, and memory complexity. The number of rounds is the maximal number of times any party must wait for information from other parties in order to proceed (this includes sequential rounds that a party may not be directly involved in). The number of public values is the maximal number of public values (keys, etc.) computed and multicast per party (without multiplicity). The communication complexity is the maximal number of broadcast/multicast messages received by any party in one call of the protocol. The memory complexity is the maximal number of stored values needed to compute the session key.

In this chapter, unlike the previous two, we have assumed a binary double-tree, meaning that parties in both compilers have a maximum possible complexity $O(\log_2 n)$. Table 5.2 shows comparable parameters for our two GKE-Cs.

Table 5.2: Comparison of our GKE compilers. $\mathfrak{A}$ denotes a value from the underlying 0/1-IKE while $\mathfrak{G}$ denotes a value from the GKE-C.

Compiler	**Rounds**	**Public**	**Comm.s**	**Memories**
GKE-C	$2\mathfrak{A}+\mathfrak{G}$	$2\mathfrak{A}+2\mathfrak{G}$	$5\mathfrak{A}+(\log_2 n)\mathfrak{G}$	$\mathfrak{A}+(\log_2 n)\mathfrak{G}$
P2P-GKE-C	$2\mathfrak{A}+(\log_2 n)\mathfrak{G}$	$2\mathfrak{A}+2\mathfrak{G}$	$5\mathfrak{A}+\mathfrak{G}$	$\mathfrak{A}+\mathfrak{G}$

Our GKE-C consists of three rounds. In the first round and second round, the 0/1-IKE public keys are exchanged, while in the third round group element products are multicast. The public values multicast by each party consist of the 0/1-IKE public key(s) and one group element product per child (or no product, in the case of the leaves). The multicast values received are the one 0/1-IKE public key of the parent and one for each child, as well as one group element product from each ancestor (minus itself, plus either $\mathcal{P}_0$ or $\mathcal{P}_1$), which is maximally $\log_2 n$, for a leaf. For the session key computation, each party requires one group element product from each ancestor (minus itself, plus either $\mathcal{P}_0$ or $\mathcal{P}_1$) as well as the 0/1-IKE key shared with the parent.

Our P2P-GKE-C, more or less, exchanges the number of rounds with the communication and memory complexity.

5.6 CONCLUSION

We introduced definitions of generic two-party KEs, with up to one interaction, and GDH problems for (G)KEs. We then proposed our GKE compiler (GKE-C) based on these generic definitions and showed how the security of our compiler reduces to the security of the underlying KE, which we assumed relied on the hardness of an underlying decisional problem. We also proposed a peer-to-peer GKE compiler (P2P-GKE-C).

We would like to again note that both the GKE-C and P2P-GKE-C can be modified to have multiple children per node, e.g. $l > 2$ (giving $O(\log_l(n))$-complexity), but also modified to have a variable number of children per node, e.g. 2 for party $\mathcal{P}_2$ and 3 for party $\mathcal{P}_3$. What this means is that the tree can be built taking into consideration the computational power and memory capacity of individual nodes. However, it must be kept in mind that the compilers essentially turn a two-party ephemeral key into a static key for the duration of the GKE. This means that, as the security of the GKE relies entirely on

the security of the underlying KE, static key vulnerabilities may be transferred to the GKE.[5]

The astute reader may question the lack of a generalization of BDI (Defn. 20) into another GKE compiler. Unfortunately BDI does not derive its security directly from the underlying KE and its hard problem, but rather a hard problem that must consider products (or compositions) of shared keys. Based on the literature so far [3, 13, 27, 60], it seems that each PQC secure BDI instantiation requires the definition of a new and specific problem for the GKE security, instead of relying on an already defined hard problem. Lastly, BDI session key generation relies on the rather limiting requirement that the underlying group be commutative, which we do not require for GKE-C, only a group.

We sincerely hope that the generality of our definitions and compilers inspires more research and publications towards generalizing current works and creating a system of cryptographic protocols such that regardless of the computational power paradigm that may come, we have a ready-set system of cryptography, ready to go.

5 As SIDH is vulnerable to the static key adaptive attack of Galbraith et al. [28], SIT is as well. However, as the attack relies on multiple successful key exchanges with non-random keys, each successful key exchange giving away 1 bit of information about a secret key, the attack is limited by the number of children per node. If even a single bit of the secret key of either $\mathcal{P}_0$ or $\mathcal{P}_1$ was known, their shared key might be discernible from random and thereby also the group shared key. An easy fix is simply to limit the number of children for $\mathcal{P}_0$ and $\mathcal{P}_1$ to one each (other than each other).

6

GENERALIZED EVEN-MANSOUR

6.1 INTRODUCTION

In 1997, Even and Mansour [24] introduced and proved security for the DES inspired block cipher scheme we now call the EM scheme: Given a public random permutation, P, over n-bit strings, with two different, random, secret, n-bit subkeys k_1 and k_2, a plaintext $x \in \{0,1\}^n$ could be enciphered as

$$EM^P_{(k_1,k_2)}(x) = P(x \oplus k_1) \oplus k_2,$$

with an obvious decryption using the inverse public permutation. Their scheme was minimal in the sense that it is necessary to XOR a key before and after the permutation, otherwise the single key may easily be found. As an improvement, Dunkelman, Keller, and Shamir [22] showed that even with identical keys the scheme would be indistinguishable from a random permutation, i.e. it was a pseudorandom permutation.

Eventually in 2012, Kuwakado and Morii [44] showed that the EM scheme could be broken by a quantum adversary with quantum queries. Rather than discard the EM construction entirely, Alagic and Russell [1] considered whether it was possible to define the two-key EM scheme over other groups in order to retain security against quantum adversaries with quantum queries.

Group actions (Defn. 10) are general algebraic structures of which groups and their group operations are special cases. Recently, group actions have come into vogue due to their usage in the post-quantum commutative supersingular isogeny Diffie-Hellman key exchange (CSIDH) of Castryck et al [11], based on work by Couveignes [16] as well as Rostovtsev and Stolbunov [56]. See Chapter 3, Section 3.3.1 for a short introduction to CSIDH.

In this dissertation, we show that given a regular group action (see Definition 10 for this definition) and a public random permutation on the underlying set, the EM permutation is indistinguishable from a random permutation to a computationally unbounded adversary, having access to only polynomially-many classical queries to its oracles. We also give evidence that for certain group actions (those with no group structure on the set), the Kuwakado and Morii attack fails.

6.2 PRELIMINARIES

In this chapter, we assume that the (probabilistic) adversary $\mathcal{A}$ is unbounded computationally, but may only make polynomially-many queries to the oracles involved, where all oracles act as classical query black-boxes. We assume the existence of efficiently computable permutations of set elements. We also assume that all algebraic groups specified are finite.

6.3 RELATED WORK

In extension of their simplification of the EM scheme, Dunkelman, Keller, and Shamir [22] attacked the construction using variants of slide attacks in order to show that the security bound was optimal. These slide attacks make use of the commutative property of the underlying group. They further considered other variants of the EM scheme, such as the "addition Even-Mansour with an involution as the permutation" (two-keyed) version. Kilian and Rogaway [42] were also inspired by EM to define their FX construction, of which the EM scheme is a special case.

As referred to in the introduction, if able to query their oracle with a superposition of quantum states, Kuwakado and Morii [44] are able to break the EM scheme on n-bit strings, using Simon's algorithm. Kaplan et al. [39], using Kuwakado and Morii's results, showed how to break many classical cipher schemes, which in turn incited Alagic and Russell [1].

6.4 GROUP ACTION EVEN-MANSOUR

We now define our generalization of the EM scheme. Essentially, we replace the XOR operation with a group action.

We define the group action Even-Mansour (GAEM) scheme to be the triple of a key generation algorithm, encryption algorithm, and decryption algorithm.

The key generation algorithm takes as input a security parameter 1^λ, outputs a group G and a set X such that G acts on X as a regular group action, as well as a key $k = (k_1, k_2)$ for $k_1, k_2 \xleftarrow{R} G$. We let P be given as a common and publicly known random permutation on X and we let P^{-1} be its inverse. It is assumed that, for any given $x \in X$, it is easy (i.e. polynomial-time) to get $P(x)$ and $P^{-1}(x)$ by using an easily and commonly accessible classical query black-box oracle.

Definition 77 (Group action Even-Mansour (GAEM))**.** *The encryption algorithm $Enc_k(m)$ takes as input the key k and a set element $m \in X$ as plaintext, and outputs*

$$Enc_k(m) = k_2 \star P(k_1 \star m).$$

The decryption algorithm $Dec_k(c)$ takes as input the key k and a set element $c \in X$ as ciphertext, and outputs

$$Dec_k(c) = k_1^{-1} \star P^{-1}(k_2^{-1} \star c).$$

This definition satisfies correctness.

We will exclude the key subscript in Enc_k where applicable.

As a remark on the public random permutation: Since the set X is finite (because of the bijection that follows from the group action and G is assumed finite), the public random permutation must be a random permutation over a finite set.

We present our GAEM pseudorandomness result as the following theorem.

Theorem 78. *For security parameter 1^λ, let a group G and a set X be given such that G acts on X as a regular group action. Assume P is a random permutation on X and let the key be $k = (k_1, k_2)$ for $k_1, k_2 \xleftarrow{R} G$. For any adversary $\mathcal{A}$, limited to polynomially-many Enc- and P/P^{-1}-oracle queries,[1] the adversarial advantage of $\mathcal{A}$, $Adv(\mathcal{A})$, is bounded by*

$$Adv(\mathcal{A}) \stackrel{\text{def}}{=} \left| Pr\left[\mathcal{A}_{Enc}^{P,P^{-1}}(1^\lambda) = 1\right] - Pr\left[\mathcal{A}_{\pi}^{P,P^{-1}}(1^\lambda) = 1\right] \right| = O\left(\frac{st}{|G|}\right), \tag{6.1}$$

where π is a random permutation on X, s is the number of Enc_k-queries and t is the number of P/P^{-1}-queries, i.e. the success probability is negligible.

Proof. We may assume that $\mathcal{A}$ is deterministic (in essence, being unbounded computationally affords $\mathcal{A}$ the possibility of derandomizing its strategy by searching all its possible random choices and picking the most effective choices after having computed the effectiveness of each choice. For an example, see [21].) Letting S_i and T_i be the sets of i *Enc*- and P/P^{-1}-queries, respectively, we may also assume that $\mathcal{A}$ never queries a pair in S_s or T_t (the final query transcripts) more than once.

Let us define two main games that $\mathcal{A}$ could play through oracle interactions, **Game R** and **Game X** (see page 87 for the explicit game descriptions). Intuitively, **Game X** behaves like **Game R** except that **Game X** checks for consistency as it does not want $\mathcal{A}$ to win on some collision. Neither game is exactly the game that would be played in (6.1), but we wish to show that the probability of an adversary winning in **Game R** or **Game X** is equivalent to the corresponding probability in (6.1).

Note that the steps in italics have no impact on the response to $\mathcal{A}$'s queries, we simply continue to answer the queries and only note if

1 The adversary is allowed access to the black-box oracles in the following way: Upon a classical query of an element $x \in X$, return $Enc_k(x) = k_2 \star P(k_1 \star x)$, $P(x)$, or $P^{-1}(x)$, depending on which oracle is queried.

the key turns *bad*, i.e. we say that a key-pair $k = (k_1, k_2)$ is **bad w.r.t. the sets** S_s **and** T_t if there exist i, j such that either $k_1 \star m_i = x_j$ or $k_2^{-1} \star c_i = y_j$, and k is **good** otherwise. There are at most $2st - |G|$ bad keys.

Game R: We consider the random game that corresponds to the latter probability in (6.1), i.e.

$$Pr\left[\mathcal{A}_{\pi}^{P,P^{-1}}(1^{\lambda}) = 1\right]. \tag{6.2}$$

As we are simply giving uniformly random answers to each of $\mathcal{A}$'s queries in **Game R**, the probability in (6.2) is equal to the probability of the adversary winning when playing **Game R**, i.e. letting Pr_R denote the probability when playing **Game R**,

$$Pr_R\left[\mathcal{A}_{Enc}^{P,P^{-1}}(1^{\lambda}) = 1\right] = Pr\left[\mathcal{A}_{\pi}^{P,P^{-1}}(1^{\lambda}) = 1\right]. \tag{6.3}$$

Game X: Note that for **Game X**, the parts in italics have no impact on the response to $\mathcal{A}$'s queries, however, when a key becomes bad, we choose a new random value repeatedly for the response until the key is no longer bad, and then reply with this value. **Game X** is much like **Game R** in choosing random values as answers, however the behavior just outlined "forces" consistency for query answers if an inconsistency was about to be created. **Game X** also does not quite correspond to the game in the former probability given in (6.1) but their probabilities are indeed equal as the following lemma shows.

Lemma 79. *Letting Pr_X denote the probability when playing **Game X**,*

$$Pr_X\left[\mathcal{A}_{Enc}^{P,P^{-1}}(1^{\lambda}) = 1\right] = Pr\left[\mathcal{A}_{Enc}^{P,P^{-1}}(1^{\lambda}) = 1\right]. \tag{6.4}$$

Proof. We begin by defining **Game X′** (see page 88). Notice that the only difference between the game defining the former probability in (6.1) and **Game X′** is that the former has defined all values for the oracles beforehand while the latter "defines as it goes." Thus, their probabilities are equal.

What we wish to show is that no adversary $\mathcal{A}$ may distinguish between playing **Game X** and playing **Game X′**. We will do this by showing that no adversary $\mathcal{A}$ may distinguish between the outputs given by the two games. As both games begin by choosing a uniformly random key $k = (k_1, k_2)$ and as we show that for this value the games are identical, we hereby assume such a key k to be a fixed, but arbitrary, value for the remainder of this lemma's proof.

Considering the definitions of **Game X** and **Game X′**, we see that the two games define their *Enc*- and P/P^{-1}-oracles differently: the former defining both, while the latter defines only the P/P^{-1}-oracle and computes the *Enc*-oracle answer. We show that **Game X** also answers its *Enc*-oracle queries by referring to P/P^{-1}, although not directly.

Notation: We let $S_i^1 = \{m|(m,c) \in S_i\}$, $S_i^2 = \{c|(m,c) \in S_i\}$, $T_i^1 = \{x|(x,y) \in T_i\}$, and $T_i^2 = \{y|(x,y) \in T_i\}$.

GAME R: Initially, let S_0 and T_0 be empty and flag unset. Choose $k = (k_1, k_2)$ for $k_1, k_2 \xleftarrow{R} G$, then answer the $i+1$-st query as follows:	**GAME X:** Initially, let S_0 and T_0 be empty and flag unset. Choose $k = (k_1, k_2)$ for $k_1, k_2 \xleftarrow{R} G$, then answer the $i+1$-st query as follows:
Enc*-oracle query with m_{i+1}:** **1.** Choose $c_{i+1} \xleftarrow{R} X \setminus S_i^2$. **2.** *If* $P(k_1 \star m_{i+1}) \in T_i^2$, *or* $P^{-1}(k_2^{-1} \star c_{i+1}) \in T_i^1$, *then set flag to* ***bad*.* **3.** Define $Enc(m_{i+1}) = c_{i+1}$ and return c_{i+1}.	***Enc*-oracle query with m_{i+1}:** **1.** Choose $c_{i+1} \xleftarrow{R} X \setminus S_i^2$. **2.** If $P(k_1 \star m_{i+1}) \in T_i^2$ then redefine $c_{i+1} := k_2 \star P(k_1 \star m_{i+1})$ *and set flag to* ***bad****.* Else if $P^{-1}(k_2^{-1} \star c_{i+1}) \in T_i^1$, *then set flag to* ***bad*** *and* goto Step 1. **3.** Define $Enc(m_{i+1}) = c_{i+1}$ and return c_{i+1}.
P*-oracle query with x_{i+1}:** **1.** Choose $y_{i+1} \xleftarrow{R} X \setminus T_i^2$. **2.** *If* $Enc(k_1^{-1} \star x_{i+1}) \in S_i^2$, *or* $Dec(k_2 \star y_{i+1}) \in S_i^1$, *then set flag to* ***bad*.* **3.** Define $P(x_{i+1}) = y_{i+1}$ (and thereby also $P^{-1}(y_{i+1}) = x_{i+1}$) and return y_{i+1}.	***P*-oracle query with x_{i+1}:** **1.** Choose $y_{i+1} \xleftarrow{R} X \setminus T_i^2$. **2.** If $Enc(k_1^{-1} \star x_{i+1}) \in S_i^2$ then redefine $y_{i+1} := k_2^{-1} \star Enc(k_1^{-1} \star x_{i+1})$ *and set flag to* ***bad****.* Else if $Dec(k_2 \star y_{i+1}) \in S_i^1$, *then set flag to* ***bad*** *and* goto Step 1. **3.** Define $P(x_{i+1}) = y_{i+1}$ (and thereby also $P^{-1}(y_{i+1}) = x_{i+1}$) and return y_{i+1}.
P^{-1}-oracle query with y_{i+1}: **1.** Choose $x_{i+1} \xleftarrow{R} X \setminus T_i^1$. **2.** *If* $Dec(k_2 \star y_{i+1}) \in S_i^1$, *or* $Enc(k_1^{-1} \star x_{i+1}) \in S_i^2$, *then set flag to* ***bad****.* **3.** Define $P^{-1}(y_{i+1}) = x_{i+1}$ (and thereby also $P(x_{i+1}) = y_{i+1}$) and return x_{i+1}.	**P^{-1}-oracle query with y_{i+1}:** **1.** Choose $x_{i+1} \xleftarrow{R} X \setminus T_i^1$. **2.** If $Dec(k_2 \star y_{i+1}) \in S_i^1$ then redefine $x_{i+1} := k_1 \star Dec(k_2 \star y_{i+1})$ *and set flag to* ***bad****.* Else if $Enc(k_1^{-1} \star x_{i+1}) \in S_i^2$, *then set flag to* ***bad*** *and* goto Step 1. **3.** Define $P^{-1}(y_{i+1}) = x_{i+1}$ (and thereby also $P(x_{i+1}) = y_{i+1}$) and return x_{i+1}.

Figure 6.1: Game R and Game X

Given the partial functions Enc and P in **Game X**, i.e. functions having been defined for all values up to and including the i-th query, define the partial function $\widehat{P}$ as the following. For $x \in X$,

$$\widehat{P}(x) \stackrel{\text{def}}{=} \begin{cases} P(x) & \text{if } P(x) \text{ is defined,} \\ k_2^{-1} \star Enc(k_1^{-1} \star x) & \text{if } Enc(k_1^{-1} \star x) \text{ is defined,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

GAME X′: Initially, let S_0 and T_0 be empty. Choose $k = (k_1, k_2)$ for $k_1, k_2 \xleftarrow{R} G$, then answer the $i+1$-st query as follows:

Enc-**oracle query with** m_{i+1}**:**
1. If $P(k_1 \star m_{i+1}) \in T_i^2$ return $k_2 \star P(k_1 \star m_{i+1})$.
2. Else choose $y_{i+1} \xleftarrow{R} X \setminus T_i^2$, define $P(k_1 \star m_{i+1}) = y_{i+1}$, and return $k_2 \star y_{i+1}$.

P-**oracle query with** x_{i+1}**:**
1. If $P(x_{i+1}) \in T_i^2$, return $P(x_{i+1})$.
2. Else choose $y_{i+1} \xleftarrow{R} X \setminus T_i^2$, define $P(x_{i+1}) = y_{i+1}$, and return y_{i+1}.

P^{-1}-**oracle query with** y_{i+1}**:**
1. If $P^{-1}(y_{i+1}) \in T_i^1$, return $P^{-1}(y_{i+1})$.
2. Else choose $x_{i+1} \xleftarrow{R} X \setminus T_i^1$, define $P^{-1}(y_{i+1}) = x_{i+1}$, and return x_{i+1}.

Figure 6.2: Game X′

Using the above definition for $\widehat{P}$, we see that defining a value for *Enc* or P implicitly defines a value for $\widehat{P}$. The first question is, whether or not $\widehat{P}$ is well-defined, i.e. whether there are clashes of values for some x for which both $P(x)$ and $Enc(k_1^{-1} \star x)$ are defined.

Lemma 80. *Let Enc and P be partial functions arising in* ***Game X****, then the partial function* $\widehat{P}$ *is well-defined.*

Proof. Proof by induction on the number of "Define" steps in **Game X** (i.e. steps $Enc-3, P-3$, and $P^{-1}-3$, see page 87) as these are the steps where $\widehat{P}$ becomes defined. The initial case of the induction proof is trivial as S_0 and T_0 are empty such that no values may clash. Suppose now that in step $Enc-3$ we define $Enc(m) = c$. The only possibility that $\widehat{P}$ becomes ill-defined will occur if the new $Enc(m)$ value clashes with a prior defined $P(k_1 \star m)$ value: If $P(k_1 \star m)$ was not defined, then no clashes can arise. If $P(k_1 \star m)$ was defined, then by step $Enc-2$, the value is $k_2^{-1} \star Enc(m)$, such that there is no clash.

Analogously, for P and P^{-1}, no clashes will arise, hence $\widehat{P}$ must be well-defined. □

We may also consider $\widehat{P}$ in **Game X′**, in the sense that when we define a value for P in the game, we implicitly define a value for $\widehat{P}$ where $\widehat{P}(x) = P(x)$ as $Enc(k_1^{-1} \star x) = P(x)$ in **Game X′**.

We wish now to show that the oracle query answers of *Enc*, P, and P^{-1} in **Game X**, expressed in terms of $\widehat{P}$, correspond exactly to those in **Game X′**, i.e. we want to show that their outputs are equivalent in

terms of $\widehat{P}$. We will also drop subscripts for S_i and T_i for the remainder of this proof.

Case 1: *Enc*-oracle query. Beginning with **Game X**, we first note that **Game X** never defines $Enc(m)$ unless m has been queried to the *Enc*-oracle. However, as $\mathcal{A}$ never repeats a query if it can guess the answer, i.e. never re-queries any *Enc*-oracle message, we may assume that $Enc(m)$ is undefined when m is queried. Therefore, we see that concurrently with m being queried, we have that $\widehat{P}(k_1 \star m)$ will be defined if and only if $P(k_1 \star m)$ is defined, and if defined then $\widehat{P}(k_1 \star m) = P(k_1 \star m)$. Let us consider the two cases: when $\widehat{P}(k_1 \star m)$ is defined and when it is undefined.

case 1a: When $\widehat{P}(k_1 \star m)$ is defined, **Game X** returns $c = k_2 \star \widehat{P}(k_1 \star m)$. Setting $Enc(m) = c$ leaves $\widehat{P}$ unchanged, i.e. the value $\widehat{P}(k_1 \star m)$ remains the same, unlike the next case.

case 1b: When $\widehat{P}(k_1 \star m)$ is undefined, **Game X** repeatedly chooses $c \xleftarrow{R} X \setminus S^2$ uniformly until $P^{-1}(k_2^{-1} \star c)$ is undefined, i.e c is in the set $U = \{c \in X | P^{-1}(k_2^{-1} \star c) \notin T^1\}$. From the definition of $\widehat{P}$ it follows that $y = k_2^{-1} \star c$ is uniformly distributed over $X \setminus \widehat{T}^2$.[2] In this case, setting $Enc(m) = c = k_2 \star y$ also sets $\widehat{P}(k_1 \star m) = y$.

We now consider the same query on **Game X′**.

case 1a′: When $\widehat{P}(k_1 \star m) = P(k_1 \star m)$ is defined, $c = k_2 \star P(k_1 \star m) = k_2 \star \widehat{P}(k_1 \star m)$ is returned, and $\widehat{P}$ is unchanged.

case 1b′: When $\widehat{P}(k_1 \star m)$ is undefined such that also $P(k_1 \star m)$ is undefined, we choose $y \xleftarrow{R} X \setminus T^2 = X \setminus \widehat{T}^2$, where $\widehat{T}^2$ is the set $\{y | \exists x \text{ s.t. } \widehat{P}(x) \text{ is defined and } \widehat{P}(x) = y\}$. $P(k_1 \star m)$ is set to y such that also $\widehat{P}(k_1 \star m) = y$, and $c = k_2 \star y$ is returned.

Thus, the behavior of **Game X** and **Game X′** are identical on the *Enc*-oracle queries.

---

2 For a proof of this statement, note that $\widehat{T}^1$ and $\widehat{T}^2$ are the corresponding sets to T^1 and T^2 on the query pairs of $\widehat{P}$. We must first show that $S^2 \cup U^{\complement} = k_2 \star \widehat{T}^2$:
"$\supseteq$": Assume that $c := k_2 \star y \in k_2 \star \widehat{T}^2$ for some $y \in \widehat{T}^2$, then either $y \in T^2$ or $k_2 \star y \in S^2$ per definition of $\widehat{P}$. If $y \in T^2$, then $\exists x \in T^1$ s.t. $P^{-1}(y) = x \in T^1 \Leftrightarrow P^{-1}(k_2^{-1} \star c) \in T^1 \Leftrightarrow c \in U^{\complement} \Rightarrow k_2 \star y \in U^{\complement}$. If $k_2 \star y \in S^2$, we are done.
"$\subseteq$": If $c \in S^2$, then $\exists m \in S^1$ s.t. $Enc(m) = c$. This means that $\widehat{P}$ is defined for $x = k_1 \star m$. Hence, $\exists y \in \widehat{T}^2$ s.t.

$$k_2 \star y = k_2 \star \widehat{P}(x) = k_2 \star k_2^{-1} \star Enc(k_1^{-1} \star x) = k_2 \star k_2^{-1} \star Enc(m) = c,$$

s.t. $c \in k_2 \star \widehat{T}^2$. If $c \in U^{\complement}$, then $P^{-1}(k_2^{-1} \star c) \in T^1$, i.e. $\exists x$ s.t. $P(x) = k_2^{-1} \star c$. As $P(x)$ is thereby defined, we have that $P(x) = \widehat{P}(x)$ s.t.

$$\widehat{P}(x) = P(x) = k_2^{-1} \star c \Leftrightarrow c = k_2 \star \widehat{P}(x) \in k_2 \star \widehat{T}^2.$$

Picking $c \xleftarrow{R} X \setminus (S^2 \cup U^{\complement})$ uniformly at random is therefore the same as picking $c \xleftarrow{R} X \setminus (k_2 \star \widehat{T}^2)$ uniformly at random. We may thus infer that $y = k_2^{-1} \star c \xleftarrow{R} X \setminus \widehat{T}^2$ is picked uniformly at random.

GAME R′: Initially, let S_0 and T_0 be empty and flag unset. Answer the $i+1$-st query as follows:

Enc-**oracle query with** M_{i+1}**:**
1. Choose $c_{i+1} \xleftarrow{R} X \setminus S_i^2$.
2. Define $Enc(m_{i+1}) := c_{i+1}$ and return c_{i+1}.

P-**oracle query with** x_{i+1}**:**
1. Choose $y_{i+1} \xleftarrow{R} X \setminus T_i^2$.
2. Define $P(x_{i+1}) := y_{i+1}$ and return y_{i+1}.

P^{-1}-**oracle query with** y_{i+1}**:**
1. Choose $x_{i+1} \xleftarrow{R} X \setminus T_i^1$.
2. Define $P^{-1}(y_{i+1}) := x_{i+1}$ and return x_{i+1}.

After all queries have been answered, choose $k = (k_1, k_2)$ *for* $k_1, k_2 \xleftarrow{R} G$. *If there exists* $(m, c) \in S_s$ *and* $(x, y) \in T_t$ *such that* k *becomes bad then set flag to* ***bad****.*

Figure 6.3: Game R′

The arguments for the cases of P and P^{-1} can be treated in a likewise manner, which we therefore skip. We conclude that the behavior of **Game X** and **Game X′** are identical on the oracle queries. Hence, an adversary cannot distinguish between the two games and the probabilities must be equivalent. □

Returning to the main proof, we have defined **Game R** and **Game X** in such a way that their outcomes differ only in the event that a key turns bad. Thus, any circumstance that causes a difference in the instructions carried out by the games, will also cause both games to set the flag to bad. Let BAD denote the event that the flag gets set to bad and the case that the flag is not set to bad by $\neg BAD$, then the two following lemmas follow from the previous statement.

Lemma 81. $Pr_R[BAD] = Pr_X[BAD]$ *and* $Pr_R[\neg BAD] = Pr_X[\neg BAD]$.

Lemma 82. $Pr_R\left[\mathcal{A}_{Enc}^{P,P^{-1}} = 1 | \neg BAD\right] = Pr_X\left[\mathcal{A}_{Enc}^{P,P^{-1}} = 1 | \neg BAD\right]$.

The following lemma then follows by using (6.3), (6.4), and lemmas 81 and 82.

Lemma 83. $Adv(\mathcal{A}) \leq Pr_R[BAD]$.

Let us define yet another game, **Game R′** (see page 90). This game runs as **Game R** except that it does not choose a key until all of the

queries have been answered and then checks whether or not the key has become bad.

Lemma 84. $Pr_R[BAD] = Pr_{R'}[BAD]$.

Proof. We need to show that the flag is set to bad in **Game R** if and only if the flag is set to bad in **Game R′**, which we do by first introducing the following definition.

Definition 85. *We say that two Enc-pairs (m_i, c_i) and (m_j, c_j) **overlap** if $m_i = m_j$ or $c_i = c_j$. If $m_i = m_j$ and $c_i = c_j$, we say that the pairs are **identical**. Likewise for P/P^{-1}-pairs (x_i, y_i) and (x_j, y_j).*

If two pairs overlap, then by the definition of the *Enc*- and P/P^{-1}-oracles, they must be identical.

"⇒": We want to show that there exists $(m, c) \in S_s$ and $(x, y) \in T_t$ such that either $k_1 \star m = x$ or $k_2^{-1} \star c = y$ (i.e. such that k becomes bad). We have to consider the 6 cases where the flag is set to bad. All of the cases use an analogous argument to the following: If $P(k_1 \star m)$ is defined then $P(k_1 \star m) = y = P(x)$ for some $(x, y) \in T_t$ such that, as overlapping pairs are identical, $k_1 \star m = x$.

"⇐": We assume that there exists $(m, c) \in S_s$ and $(x, y) \in T_t$ such that k becomes bad. i.e. such that either $k_1 \star m = x$ or $k_2^{-1} \star c = y$. We need to check that in all three oracle queries, the flag in **Game R** is set to bad, which needs a consideration of 6 cases.

Assume that $k_1 \star m = x$, then

$$\begin{aligned} \textit{Enc}\text{-oracle on } m &: P(k_1 \star m) = P(x) = y \in T_t^2, \\ P\text{-oracle on } x &: \textit{Enc}(k_1^{-1} \star x) = \textit{Enc}(m) = c \in S_s^2, \\ P^{-1}\text{-oracle on } y &: \textit{Enc}(k_1^{-1} \star P^{-1}(y)) = \textit{Enc}(k_1^{-1} \star x) \\ &\quad = \textit{Enc}(m) = c \in S_s^2. \end{aligned}$$

Assume now that $k_2^{-1} \star c = y$, then

$$\begin{aligned} \textit{Enc}\text{-oracle on } m &: P^{-1}(k_2^{-1} \star c) = P^{-1}(y) = x \in T_t^1, \\ P\text{-oracle on } x &: \textit{Dec}(k_2 \star P(x)) = \textit{Dec}(k_2 \star y) \\ &\quad = \textit{Dec}(c) = m \in S_s^1, \\ P^{-1}\text{-oracle on } y &: \textit{Dec}(k_2 \star y) = \textit{Dec}(c) = m \in S_s^1. \end{aligned}$$

□

Using the above lemma, we now only have to bound $Pr_{R'}[BAD]$ in order to bound $Adv(\mathcal{A})$, but as the adversary queries at most s elements to the *Enc*-oracle, at most t elements to the P/P^{-1}-oracles, and the subkeys of $k = (k_1, k_2)$ are chosen uniformly at random from G, we have that the probability of choosing a bad key is at most $2st/|G|$, i.e.

$$Adv(\mathcal{A}) \leq Pr_{R'}[BAD] = O\left(\frac{st}{|G|}\right).$$

□

Stated simply,

Theorem 86. *For any adversary $\mathcal{A}$, limited to polynomially-many Enc- and P/P^{-1}-oracle queries, the GAEM scheme over a regular group action G acting on X, is a pseudorandom permutation.*

By adding a decryption oracle, we get the following theorem:

Theorem 87. *For any adversary $\mathcal{A}$, limited to polynomially-many Enc/Dec- and P/P^{-1}-oracle queries, the GAEM scheme over a regular group action G acting on X, is a super pseudorandom permutation.*

We remark that, as in [42], our proof also works with minor changes for the identical subkeys case, i.e. where $k_1 = k_2$.

6.4.1 *Evidence for quantum security*

There are two major questions applicable to our generalized construction of the EM scheme: 1. Is the construction secure against Kuwakado and Morii's attack in [39, 44]? and 2. Is the construction a post-quantum PRP?

Although we have not yet been able to prove the latter, leaving it for future work, for the former, we argue below in the affirmative.

6.4.1.1 *Resistance against Simon's style attacks.*

In essence, Kuwakado and Morii's attack is a differential cryptanalysis attack, i.e. by manipulating plain/ciphertexts and using the oracles, they are able to tease out a key and thereby both keys in EM. More specifically, they use the oracles to create an instance of Simon's problem.

Definition 88 (Simon's problem [39])**.** *Given a Boolean function $f : \{0,1\}^n \to \{0,1\}^n$ and the promise that there exists $s \in \{0,1\}^n$ such that for any $(x,y) \in \{0,1\}^n$, $[f(x) = f(y)] \Leftrightarrow [x \oplus y \in \{0^n, s\}]$, the goal is to find s.*

There is a quantum algorithm (Simon's algorithm) that solves this problem with quantum complexity $O(n)$ (see Kaplan et al. [39] for more details). To explain Kuwakado and Morii's attack, Kaplan et al. use the EM oracles to define the function,

$$\begin{aligned} f : \{0,1\}^n &\to \{0,1\}^n \\ x \quad &\mapsto E_{(k_1,k_2)}(x) \oplus P(x) = P(x \oplus k_1) \oplus k_2 \oplus P(x). \end{aligned} \tag{6.5}$$

This function obviously satisfies the promise in Simon's problem with $s = k_1$ and so it may be solved. We call such an attack a *Simon style attack*.

In the case of GAEM, unless X has a well-defined group structure, there is no operation defined on X as it is simply a set, hence we cannot define f as in (6.5). Assuming X is such a set, the only functions we can define on it, using the given oracles, are through composition and using the group action. However, such functions, being compositions of bijective functions, will be bijective, such that the condition on the function in Simon's problem can never be satisfied as there cannot exist a function with a non-trivial shift s. Thus, a Simon style attack cannot work directly on GAEM when X is not a group.

6.5 CONCLUDING REMARKS

In this chapter, we generalized the Even-Mansour scheme to regular group actions and proved that classical results pertain to the group action version.

Our version is as general a construction as we might reasonably construct entirely based on group theory. We have given evidence that it is not vulnerable to the quantum attack of Kuwakado and Morii [44] when the set does not have a group structure. A proof of post-quantum security would require extensive research into how to prove quantum indistinguishability for classical constructions, which is the next logical step in our research.

Although our construction has yet to be proved post-quantum secure, we hope that it opens avenues for classical schemes to be generalized so that we may preserve the cryptographic heritage from many decades of research.

Part III

CONCLUSION

7

CONCLUDING REMARKS

7.1 SUMMARY

In this dissertation, we gave the mathematical and cryptographical background needed to understand both isogeny based and lattice based approaches to post-quantum cryptography relevant key exchanges and group key exchanges as well as for pseudorandom permutations. We then defined and proved security for isogeny based and lattice based group key exchanges, both a concurrent and a sequential version. We furthermore gave compilers turning these group key exchanges into authenticated group key exchanges by using a signature scheme to sign all messages between parties. Furthermore, we considered the generalization of the underlying key exchanges and their hard problems in order to generalize the group key exchanges themselves, resulting in a GKE compiler that only relies on the form of the underlying two-party key exchange and its associated hard problem. Finally, we generalized the Even-Mansour pseudorandom permutation from bit strings to group actions and proved the indistinguishability against arbitrary adversaries having only a polynomial number of classical oracle queries.

7.2 DISCUSSION

At present, PQC is quite a new field within modern cryptography and modern cryptography is itself a rather new field in applied mathematics. Modern cryptography has seen increasingly more usage thanks to the invention of the internet and the consequent need for information security in the public, private, and governmental sectors. The cause is the quandary of the information age: the ability to decide when and where information is shared. We not only need to be able to provide security, we also need to be able to deliver on speed and size, i.e. efficiency. The GKEs presented in this dissertation deliver that by being an entire order of magnitude more efficient than the best alternatives making them highly competitive PQC (A)GKE candidates.

Both BDII and EM are relatively old ideas, both first published in 1997. They also build on fundamental cryptographic ideas, two-party KEs and pseudorandom permutations, respectively. That we can adapt these constructions to newer/more general ideas is not entirely surprising. The structures they build on are already quite general and their original purpose was to make minimal schemes. However, the focus of cryptography generally seems to be to work with what exists.

It makes sense to only consider existing constructions and build from them as they are usually already implemented and widely in use. It is also easier to get published when there is an incentive behind the work, such as improving designs or giving further uses for widely used constructions. Mathematics too used to be focused on practical applications, but mathematics did not truly become the giant it is today until it became the standard to abstract it. This abstraction fed back into the real world applications, indeed in many fields. We therefore aim to abstract cryptographic constructions, and cryptography in general, in the hopes that they too can lead to such a feedback loop.

Creating a system of generic cryptographic constructions that only rely on hard problems also frees cryptographers to focus on attempting to create and break hard problems. The more work that is put into breaking hard problems, the more secure they become, is the mode of thought in cryptography. Furthermore, the more hard problem candidates there are, the better the chance of having some that are secure against even the most outlandish of adversaries. Hence, we hope that the aim of this dissertation sees a trend growing in the wider cryptographic community, namely that of generalization and creating a generic framework from basic building blocks. The more fundamental and generic we can make cryptography, the more obvious the flaws and strengths will become. This is not to say that specialized constructions are unneeded, just that a framework will benefit cryptography greatly in the long-term.

We also hope that our results can be expanded upon, implemented, and even improved. Although we have shown that a generic GKE compiler exists, we have not shown that it is minimal. Depending on the underlying two-party KE, there may even be ways to improve the resulting GKE by integrating mathematical properties inherent in the KE. There may also be better AGKE alternatives that integrate the signature scheme in a different way, or get rid of it entirely. If other parts of cryptography become generalized we may have more constructions to work with, leading into a cycle of improvement. Our constructions are also fundamental enough that there is hope that other fundamental constructions, and the constructions that build on them, will be generalizable as well.

7.3 FUTURE WORK

Although we were able to improve the complexity of the best PQC GKE and AGKE candidates, and we were able to generalize the structure of both our GKEs and the EM scheme, we were not able to make the GKEs and AGKEs more efficient than logarithmic communication and memory complexity, with a small constant time trade-off to the round complexity. We were not able to improve the security models such that the GKEs and AGKEs were secure in the strong-corruption model nor

the EM scheme against adversaries with polynomially-many quantum queries.

As for future work in extension of the GKEs presented in this dissertation, the first priority should go to proving security in the G-CK$^+$ model because the stronger the security model, the wider the applicability of the construction. We may have to alter our GKEs substantially, in which case we will need to consider the cost of doing so. The priority should be to maintain the logarithmic complexity, and for the isogeny based versions, as the isogeny computation is generally slow so we would hope that such an alteration will not add to the number of isogeny computations performed.

The work of Choi et al. [13] also inspire us to try to integrate the structure of the KEs if possible, as those authors have done by integrating the R-LWE KE into the Dutta-Barua GKE [23]. If possible, we would seek to do so with both SIDH and R-LWE.

Our work is largely theoretic so an obvious next step would be to create implementations and do implementation analysis for all our GKEs. A choice of signature scheme would be required for this step, which would require an extensive review of the literature. There may also be possibilities of integrating the signature schemes further with the AGKEs, depending on which is chosen. Implementations would also require us to consider parameters in detail, giving us specific security bounds.

Trust in and application of our GKE compiler can also be improved by showing how other KEs fit into our framework and definitions. That is only a start of course, as the generalization work that we propose in this dissertation would require an extensive review of all cryptographic constructions.

As for our EM generalization, future work would include an implementation with some PRP. If we could create a PQC secure PRP candidate and show security, then such a PRP would be ideal for an implementation. Promising frameworks have been proposed independently by Alamati-De Feo-Montgomery-Patranabis [2], Moriya-Onuki-Takagi [50], and Boneh-Kogan-Woo [6] at Asiacrypt 2020, giving PQC secure pseudorandom functions from group actions. We hope that PRPs from group actions are not far behind.

Our security proof for GAEM is perfectly fine for classical security, but proving security for an adversary with access to polynomially-many *quantum* oracle queries, i.e. PQC security, is still an open problem. This should also be a priority for future research.

Part IV

APPENDIX

A

TABLE OF KEY EXCHANGES IN THE 0/1-IKE NOTATION AND THEIR CORRESPONDING HARD PROBLEMS

Key exchanges			
	DH [19]	**SIDH** [25]	**R-LWE KE** [20, 52]
0/1	0	0	1
$\mathfrak{P}_0$	$(\mathbb{G}, q, g)$	$(p, E, \{P_0, Q_0\}, \{P_1, Q_1\})$	$(n, m, R, q, R_q, \chi, a)$
$\mathfrak{P}_1$	Same as $\mathfrak{P}_0$	$(p, E, \{P_1, Q_1\}, \{P_0, Q_0\})$	Same as $\mathfrak{P}_0$
sk_0	$x_0 \xleftarrow{R} \mathbb{Z}_q$	$r_0 \xleftarrow{R} \mathbb{Z}/\ell_0^{e_0}\mathbb{Z}$	$s_0, e_0 \xleftarrow{R} \chi$
sk_1	$x_1 \xleftarrow{R} \mathbb{Z}_q$	$r_1 \xleftarrow{R} \mathbb{Z}/\ell_1^{e_1}\mathbb{Z}$	$s_1, e_1 \xleftarrow{R} \chi$
pk_0	$h_0 = g^{x_0}$	$(E_0, \phi_0(P_1), \phi_0(Q_1))$	$b_0 = as_0 + e_0$
pk_1	$h_1 = g^{x_1}$	$(E_1, \phi_1(P_0), \phi_1(Q_0))$	$b_1 = as_1 + e_1$
sk'_A	None	None	$e'_1 \xleftarrow{R} \chi$
pk'_A	None	None	$c = \langle \bar{v} \rangle_{2q,2}$
k_I	$h_1^{x_0}$	$j(E_{0,1})$	$\mathtt{rec}(2b_1s_0, c)$
k_A	$h_0^{x_1}$	$j(E_{1,0})$	$\lceil \bar{v} \rfloor_{2q,2}$
Hard problems			
	DDH [19]	**SSDDH** [25]	**DDH-like** [7]
$\mathfrak{P}$	$(\mathbb{G}, q, g)$	$\mathfrak{P}_i = (p, E, \{P_i, Q_i\}, \{P_{i-1}, Q_{i-1}\})$ for $i = 0, 1$	(n, R, q, R_q, χ, a)
pk	(h_0, h_1)	(pk_0, pk_1)	$(b_0, (b_1, c))$
k	$g^{x_0x_1}$ *or* g^z for $z \xleftarrow{R} \mathbb{Z}_q$	$E_{0,1} \cong E/\langle P_0 + [r_0]Q_0, P_1 + [r_1]Q_1 \rangle$ *or* $E_x \cong E/\langle P_0 + [r'_0]Q_0, P_1 + [r'_1]Q_1 \rangle$ for $r'_i \xleftarrow{R} \mathbb{Z}/\ell_i^{e_i}\mathbb{Z}, i = 0, 1$	$\lceil \bar{v} \rfloor_{2q,2}$ *or* $k \xleftarrow{R} \{0,1\}^n$

In the table, we consider DH, SIDH, and R-LWE KE, from left to right. In the upper half, we note whether the KE is 0-interactive or 1-interactive and then list the protocol specific values for $\mathfrak{P}_0, \mathfrak{P}_1, sk_0, sk_1, pk_0, pk_1, sk'_A, pk'_A, k_I$, and k_A, if applicable. In the second half, we do the same for the respective hard problems DDH, SSDDH, and DDH-like.

BIBLIOGRAPHY

[1] Gorjan Alagic and Alexander Russell. "Quantum-Secure Symmetric-Key Cryptography Based on Hidden Shifts." In: *EUROCRYPT (3)*. Vol. 10212. LNCS. 2017, pp. 65–93. URL: https://arxiv.org/abs/1610.01187.

[2] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. "Cryptographic Group Actions and Applications." In: *Advances in Cryptology - ASIACRYPT 2020*. Ed. by S. Moriai and H. Wang. Vol. 12492. LNCS. Springer, Cham, 2020, pp. 411–439. DOI: 10.1007/978-3-030-64834-3_14.

[3] Daniel Apon, Dana Dachman-Soled, Huijing Gong, and Jonathan Katz. "Constant-Round Group Key Exchange from the Ring-LWE Assumption." In: *PQCrypto*. Springer, 2019, pp. 189–205.

[4] Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. "Practical Supersingular Isogeny Group Key Agreement." In: *IACR Cryptol. ePrint Arch.* (2019).

[5] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records." In: *Public Key Cryptography - PKC 2006*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Springer Berlin Heidelberg, 2006, pp. 207–228.

[6] Dan Boneh, Dimitry Kogan, and Katharine Woo. "Oblivious Pseudorandom Functions from Isogenies." In: *Advances in Cryptology - ASIACRYPT 2020*. Ed. by S. Moriai and H. Wang. Vol. 12492. LNCS. Springer, Cham, 2020, pp. 520–550. DOI: 10.1007/978-3-030-64834-3_18.

[7] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. "Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem." In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2015, pp. 553–570.

[8] Mike Burmester and Yvo Desmedt. "A secure and efficient conference key distribution system." In: *Advances in Cryptology — EUROCRYPT'94*. Ed. by Alfredo De Santis. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 275–286.

[9] Mike Burmester and Yvo Desmedt. "Efficient and Secure Conference-Key Distribution." In: *Proceedings of the International Workshop on Security Protocols*. London, UK, UK: Springer-Verlag, 1997, pp. 119–129.

[10] Ran Canetti and Hugo Krawczyk. "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels." In: *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*. EUROCRYPT '01. London, UK, UK: Springer-Verlag, 2001, pp. 453–474. ISBN: 3-540-42070-3. URL: http://dl.acm.org/citation.cfm?id=647086.715688.

[11] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action." In: *ASIACRYPT (3)*. Vol. 11274. LNCS. Springer, 2018, pp. 395–427.

[12] WhatsApp Help Center. *How to create a group*. 2022. URL: https://faq.whatsapp.com/kaios/chats/how-to-create-a-group/?lang=en.

[13] Rakyong Choi, Dongyeon Hong, and Kwangjo Kim. *Constant-round Dynamic Group Key Exchange from RLWE Assumption*. Cryptology ePrint Archive, Report 2020/035. 2020.

[14] Information Technology Laboratory Computer Security Division. *Public-key post-quantum cryptographic algorithms: Nominations*. 2016. URL: https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms.

[15] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie-Hellman." In: *CRYPTO*. Springer, 2016, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21.

[16] Jean Marc Couveignes. "Hard Homogeneous Spaces." In: *IACR Cryptology ePrint Archive* 2006 (2006), p. 291. URL: https://ia.cr/2006/291.

[17] Luca De Feo and David Jao. *defeo/sidh-paper*. URL: https://github.com/defeo/sidh-paper/blob/master/eprint.tex.

[18] Yvo Desmedt, Tanja Lange, and Mike Burmester. "Scalable Authenticated Tree Based Group Key Exchange for Ad-Hoc Groups." In: *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2007, pp. 104–118.

[19] Whitfield Diffie and Martin Hellman. "New Directions in Cryptography." In: *IEEE Trans. Inf. Theor.* 22.6 (2006), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[20] Jintai Ding, Xiang Xie, and Xiaodong Lin. *A Simple Provably Secure Key Exchange Scheme Based on the Learning with Errors Problem*. Cryptology ePrint Archive, Report 2012/688. 2012.

[21] Yan Zong Ding and Michael O. Rabin. "Hyper-Encryption and Everlasting Security." In: *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*. 2002, pp. 1–26. DOI: 10.1007/3-540-45841-7_1.

[22] Orr Dunkelman, Nathan Keller, and Adi Shamir. "Minimalism in Cryptography: The Even-Mansour Scheme Revisited." In: *EUROCRYPT*. Vol. 7237. LNCS. Springer, 2012, pp. 336–354.

[23] Ratna Dutta and Rana Barua. "Constant Round Dynamic Group Key Agreement." In: *Information Security*. Springer Berlin Heidelberg, 2005, pp. 74–88.

[24] Shimon Even and Yishay Mansour. "A Construction of a Cipher from a Single Pseudorandom Permutation." In: *J. Cryptol.* 10.3 (1997), pp. 151–162.

[25] Luca De Feo, David Jao, and Jérôme Plût. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies." In: *J. Math. Cryptol.* 8.3 (2014), pp. 209–247. DOI: 10.1515/jmc-2012-0015.

[26] Atsushi Fujioka, Katsuyuki Takashima, and Kazuki Yoneyama. "One-Round Authenticated Group Key Exchange from Isogenies." In: *Provable Security - 13th International Conference, ProvSec 2019, Cairns, QLD, Australia, October 1-4, 2019, Proceedings*. Ed. by Ron Steinfeld and Tsz Hon Yuen. Vol. 11821. Lecture Notes in Computer Science. Springer, 2019, pp. 330–338. DOI: 10.1007/978-3-030-31919-9_20.

[27] Satoshi Furukawa, Noboru Kunihiro, and Katsuyuki Takashima. "Multi-party Key Exchange Protocols from Supersingular Isogenies." In: *2018 International Symposium on Information Theory and Its Applications (ISITA)* (2018), pp. 208–212.

[28] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. "On the Security of Supersingular Isogeny Cryptosystems." In: *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. 2016, pp. 63–91. DOI: 10.1007/978-3-662-53887-6_3.

[29] Steven D. Galbraith and Frederik Vercauteren. *Computational problems in supersingular elliptic curve isogenies*. Cryptology ePrint Archive, Report 2017/774. 2017. URL: https://ia.cr/2017/774.

[30] Tetsuya Hatano, Atsuko Miyaji, and Takashi Sato. "T-robust Scalable Group Key Exchange Protocol with O(Log N) Complexity." In: *Proceedings of the 16th Australasian Conference on Information Security and Privacy*. Springer-Verlag, 2011, pp. 189–207.

[31] Hector B. Hougaard. "How to Generate Pseudorandom Permutations Over Other Groups: Even-Mansour and Feistel Revisited." In: *CoRR* abs/1707.01699 (2017). URL: http://arxiv.org/abs/1707.01699.

[32] Hector B. Hougaard and Atsuko Miyaji. "SIT: supersingular isogeny tree-based group key exchange." In: *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*. IEEE, 2020, pp. 46–53. DOI: 10.1109/AsiaJCIS50894.2020.00019.

[33] Hector B. Hougaard and Atsuko Miyaji. "Tree-Based Ring-LWE Group Key Exchanges with Logarithmic Complexity." In: *Information and Communications Security*. Ed. by W. Meng, D. Gollmann, C. D. Jensen, and J. Zhou. Vol. 12282. LNCS. Springer International Publishing, 2020, pp. 91–106. DOI: 10.1007/978-3-030-61078-4_6.

[34] Hector B. Hougaard and Atsuko Miyaji. "Authenticated logarithmic-order supersingular isogeny group key exchange." In: *International Journal of Information Security*. Springer, 2021. DOI: 10.1007/s10207-021-00549-4.

[35] Hector B. Hougaard and Atsuko Miyaji. "Authenticated tree-based R-LWE group key exchange." In: *The Computer Journal*. Ed. by Oxford Press. 2021. DOI: 10.109/comjnl/bxab165.

[36] Hector B. Hougaard and Atsuko Miyaji. "Group key exchange compilers from generic key exchanges." In: *International Conference on Network and System Security (NSS)*. LNCS. Springer International Publishing, 2021. DOI: 10.1007/978-3-030-92708-0_10.

[37] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. "Generic Compilers for Authenticated Key Exchange." In: *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*. Vol. 6477. LNCS. Springer, 2010, pp. 232–249. URL: https://www.iacr.org/archive/asiacrypt2010/6477232/6477232.pdf.

[38] David Jao and Luca De Feo. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies." In: *Post-Quantum Cryptography*. Ed. by Bo-Yin Yang. Springer Berlin Heidelberg, 2011.

[39] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. "Breaking Symmetric Cryptosystems Using Quantum Period Finding." In: *CRYPTO (2)*. Vol. 9815. LNCS. Springer, 2016, pp. 207–237.

[40] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. 2nd ed. CRC Press, 2015.

[41] Jonathan Katz and Moti Yung. "Scalable Protocols for Authenticated Group Key Exchange." In: *J. Cryptol.* 20.1 (2007), pp. 85–113.

[42] Joe Kilian and Phillip Rogaway. "How to Protect DES Against Exhaustive Key Search (an Analysis of DESX)." In: *J. Cryptol.* 14.1 (2001), pp. 17–35.

[43] Hidenori Kuwakado and Masakatu Morii. "Quantum distinguisher between the 3-round Feistel cipher and the random permutation." In: *ISIT*. IEEE, 2010, pp. 2682–2685.

[44] Hidenori Kuwakado and Masakatu Morii. "Security on the quantum-type Even-Mansour cipher." In: *ISITA*. IEEE, 2012, pp. 312–316. ISBN: 978-1-4673-2521-9.

[45] Yong Li, Sven Schäge, Zheng Yang, Christoph Bader, and Jörg Schwenk. "New Modular Compilers for Authenticated Key Exchange." In: *Applied Cryptography and Network Security*. Ed. by Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay. Cham: Springer International Publishing, 2014.

[46] Michael Luby and Charles Rackoff. "How to construct pseudorandom permutations from pseudorandom functions." In: *SIAM J. Comput.* 17.2 (1988), pp. 373–386.

[47] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "A Toolkit for Ring-LWE Cryptography." In: *Advances in Cryptology - EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Nguyen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[48] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings." In: *J. ACM* 60.6 (2013), pp. 43–78. DOI: 10.1145/2535925.

[49] Mark Manulis, Koutarou Suzuki, and Berkant Ustaoglu. "Modeling Leakage of Ephemeral Secrets in Tripartite/Group Key Exchange." In: *Information, Security and Cryptology – ICISC 2009*. Ed. by Donghoon Lee and Seokhie Hong. Springer Berlin Heidelberg, 2010, pp. 16–33.

[50] Tomoki Moriya, Hiroshi Onuki, and Tsuyoshi Takagi. "SiGamal: A Supersingular Isogeny-Based PKE and Its Application to a PRF." In: *Advances in Cryptology - ASIACRYPT 2020*. Ed. by S. Moriai and H. Wang. Vol. 12492. LNCS. Springer, Cham, 2020, pp. 551–580. DOI: 10.1007/978-3-030-64834-3_19.

[51] National Institute of Standards and Technology. *Data Encryption Standard (DES)*. FIPS Publication 46-3. 1999. URL: http://csrc.nist.gov/publications/fips/fips46-3/.

[52] Chris Peikert. "Lattice Cryptography for the Internet." In: *Post-Quantum Cryptography*. Springer International Publishing, 2014, pp. 197–219.

[53] Edoardo Persichetti, Rainer Steinwandt, and Adriana Suárez Corona. "From Key Encapsulation to Authenticated Group Key Establishment—A Compiler for Post-Quantum Primitives." In: *Entropy* 21.12 (2019). DOI: 10.3390/e21121183.

[54] John Proos and Christof Zalka. "Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves." In: *Quantum Info. Comput.* 3.4 (2003), pp. 317–344.

[55] Victoria de Quehen, Péter Kutas, Chris Leonardi, Chloe Martindale, Lorenz Panny, Christophe Petit, and Katherine E. Stange. "Improved Torsion-Point Attacks on SIDH Variants." In: *Advances in Cryptology - CRYPTO 2021*. Ed. by T. Malkin and C. Peikert. Vol. 12827. LNCS. Springer, 2021, pp. 432–470. DOI: 10.1007/978-3-030-84252-9_15.

[56] Alexander Rostovtsev and Anton Stolbunov. "Public-Key Cryptosystem Based on Isogenies." In: *IACR Cryptology ePrint Archive* 2006 (2006), p. 145.

[57] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics. Dordrecht: Springer, 2009.

[58] Vikram Singh. "A Practical Key Exchange for the Internet using Lattice Cryptography." In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 138. URL: https://eprint.iacr.org/2015/138.

[59] Koutarou Suzuki and Kazuki Yoneyama. "Exposure-Resilient One-Round Tripartite Key Exchange without Random Oracles." In: *IEICE Transactions* 97-A.6 (2014), pp. 1345–1355.

[60] Katsuyuki Takashima. "Post-Quantum Constant-Round Group Key Exchange from Static Assumptions." In: *International Symposium on Mathematics, Quantum Theory, and Cryptography*. Ed. by Tsuyoshi Takagi, Masato Wakayama, Keisuke Tanaka, Noboru Kunihiro, Kazufumi Kimoto, and Yasuhiko Ikematsu. Singapore: Springer Singapore, 2021, pp. 251–272.

[61] Qiang Tang and Chris Mitchell. "Efficient Compilers for Authenticated Group Key Exchange." In: *LNCS*. Vol. 3802. Springer, 2006, pp. 192–197.

[62] Eric Thormarker. "Post-quantum cryptography: Supersingular isogeny Diffie-Hellman key exchange." Thesis. Stockholm university. 2017.

[63] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition*. 2nd ed. Chapman & Hall/CRC, 2008. ISBN: 9781420071467.

[64] WhatsApp. *WhatsApp Encryption Overview -Technical white paper*. 2018. URL: https://www.whatsApp.com/security/WhatsApp-Security-Whitepaper.pdf.

[65] Mark Zhandry. "A Note on Quantum-Secure PRPs." In: *CoRR* abs/1611.05564 (2016). URL: http://arxiv.org/abs/1611.05564.

[66] Jiang Zhang, Zhenfeng Zhang, Jintai Ding, Michael Snook, and Özgür Dagdelen. "Authenticated Key Exchange from Ideal Lattices." In: *EUROCRYPT (2)*. Springer, 2015, pp. 719–751. URL: https://www.iacr.org/archive/eurocrypt2015/90560281/90560281.pdf.