Doctoral theses at NTNU, 2022:235

Tjerand Aga Silde

Privacy-Preserving Cryptography from Zero-Knowledge Proofs

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Information Technology and Electrical
Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Tjerand Aga Silde

Privacy-Preserving Cryptography from Zero-Knowledge Proofs

Thesis for the Degree of Philosophiae Doctor

Trondheim, August 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

ISBN 978-82-326-6571-6 (printed ver.)
ISBN 978-82-326-5212-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2022:235

Printed by NTNU Grafisk senter

Acknowledgments

You are now reading my Ph.D. thesis in cryptography, which concludes my integrated Ph.D.-program in mathematical sciences at the Department of Mathematics at the Norwegian University of Science and Technology. I am very grateful to my supervisor Kristian Gjøsteen for providing me with interesting research problems, always having an open door, being encouraging, answering all of my questions and for sharing lessons from academia and life in general. I would also like to thank my co-supervisor Colin Boyd for his support; I wish our research interest had intersected more.

Thanks to all of my collaborators Kristian, Diego F. Aranha, Carsten Baum, Thomas Haines, Johannes Muller, Peter Rønne, Martin Strand and Thor Tunge for fruitful work on designing and implementing new privacy-preserving protocols and zero-knowledge proofs, in which this thesis builds upon.

I would like to especially thank Diego and Carsten for hosting me for one week at Aarhus University in November 2019 and for four months in the fall of 2020, and Jonathan Katz for hosting me for five months at the University of Maryland during the winter 2021/22. I would also like to thank Jonathan, Andreea Alexandru, Anamaria Costache, Pia Bauspieß and Kamil Doruk Gur for taking part in exciting ongoing research that hopefully will result in new publications the coming year.

I am very grateful to be a member of the NTNU Applied Cryptology Lab, and I consider all of you both colleagues and friends. I have enjoyed the cabin trips, skiing trips, hikes, grilling in the park, game nights and parties as well as the reading groups, seminars and conference trips during the past few years. You have been an essential part of my Ph.D. life. I appreciated being a part of the training groups with Erlend Due Børve, Mayank Raikwar, Magnus Ringerud, Mattia Veroni and, when staying in Aarhus, Matteo Campanelli, Rahul Rachuri, Nikolaj Schwartzbach, Bas Spitters and Akira Takahashi. This has been crucial to ensure good mental and physical health during the long hours of conducting research these past years. I also want to thank Roger Antonsen, Bor de Kock, Jonathan Eriksen, Tom Hanson, Hunter Kippen, Kelsey Moran, Morten Solberg, Eduardo Soria-Vazquez, Jana Sotakova and Maurice Shih for all the good conversations.

I am very fortunate to work with my good colleagues in Pone Biometrics and appreciate being involved in research and development of on-marked security products, broadening my interest and knowledge about real-life cryptography and security. I am looking forward to the forthcoming journey. I also learned a lot during the very fruitful collaboration across industry and academia with Martin, Johannes Brodwall and Henrik Walker Moe when improving privacy of the Norwegian contact tracing app during the fall of 2020.

Last, but not least, I want to thank Dahlia, my family and my friends for always supporting my work, my interests and my path in life. A final thanks goes to Ana, Magnus, Mattia and Pia for proofreading my thesis.

Tjerand Aga Silde
Trondheim, May 2022

Introduction

Many real-world systems require that users are authenticated or that information is certified while keeping the identity or content secret. This sounds like a paradox, but when carefully designing new cryptographic protocols, we are able to provide practical solutions to such applications. Some recent popular examples are anonymous browsing with spam-protection [DGS$^+$18], anonymous telemetry collection [HIJ$^+$21], privacy-preserving contact-tracing [T$^+$20], anonymous broadcasting [CWF13], outsourced computation [LW18] and electronic voting [Adi08, Gjø11, LPT19], to mention but a few.

Our main tools to construct these systems are so-called *zero-knowledge proofs*. These are *cryptographic proofs* (also called *arguments*) where a prover can prove statements about secret (encrypted or committed) data to a verifier without leaking the sensitive information itself. Zero-knowledge proofs were developed by Goldwasser, Micali and Rachoff [GMR89] in 1985, and we have recently seen a dramatic increase in interest and development, pushing these protocols from a rich theory into real-world applications.

In more detail, zero-knowledge proofs have three properties. *Completeness* ensures that if the statement is true and the parties follow the protocol, the proof will be accepted. *Soundness* ensures that if the statement is false, then the verifier will not accept a proof from a cheating prover, except with negligible probability. *Zero-knowledge* ensures that the verifier learns nothing from the proof other than the fact that the statement is true. Many of the basic zero-knowledge protocols consist of three rounds of communication between a prover and a verifier. The first message from the prover is then called a *com-*

mitment, the next message from the verifier is called a *challenge*, and the final message from the prover is called a *response*. These interactive protocols can be turned into a non-interactive protocol where the challenge is obtained by hashing the statement and the commitment message, leading to a single-message proof that can be published without any interaction. This method is called the Fiat-Shamir Transform [FS87], and is applied to all (except potentially one) of the zero-knowledge protocols in this thesis.

The applications mentioned above are all built using public-key cryptography. Today, the security of public-key cryptosystems is mostly based on hard computational problems such as factoring large bi-primes [RSA78] or computing discrete logarithms over finite fields or elliptic curve groups [DH76, Mil86, Kob87]. However, Shor developed an algorithm [Sho94, Sho97] that, implemented on a large quantum computer, would efficiently solve these problems. Recent developments in quantum computing indicate that this is a threat that should be taken seriously, and over the past decade many researchers have designed cryptographic protocols that are secure against quantum computers. This is the field of post-quantum cryptography.

It is not obvious which computational problems are secure against quantum computers, which are not, and what kind of attacks we can perform by combining the joint forces of classical and quantum computers. However, some problems in the fields of lattices [Ajt96, Reg05], coding theory [McE78, Nie86], multivariate polynomials [MI88, Pat95], isogenies of elliptic curves [Sto10, JD11] and more, including constructions based purely on one-way functions [BDH11, BHH$^+$15], give rise to protocols that are seemingly secure against quantum adversaries. It is worth noting that these underlying problems have the potential to support new constructions, e.g. fully homomorphic encryption [Gen09, BGV12], that do not seem possible to build based on factoring or discrete logarithms. Hence, this area of research has great value even if a large quantum computer is never realized.

The main effort within post-quantum cryptography so far has focused on creating new schemes for key-exchange, encryption and digital signatures, to replace existing parts in established protocols such as TLS[1] or the Signal protocol[2]. These schemes are currently being standardized by NIST [AASA+20], and will be ready for real-world use from 2024 and onward[3]. However, more effort is needed to build practical cryptographic tools supporting applications such as quantum-safe anonymous browsing with spam-protection, anonymous telemetry collection, privacy-preserving contact-tracing, anonymous broadcasting, outsourced computation and electronic voting.

The main research goal of this thesis was to design new protocols based on zero-knowledge proofs for privacy applications.

The area of lattice-based zero-knowledge proofs has gained a lot of attention recently, including proofs of linear relations [BKLP15, BDL+18], amortized proofs of short pre-images [BBC+18], exact proofs of shortness [YAZ+19, BLS19, Beu20, ENS20], proofs of multiplication [BKLP15, ALS20], proofs of integer relations [LNS20] and more. These are essential building blocks and, even though useful on their own, can be used to prove more complex statements such as correct shuffle, correct decryption, or other relations. These proof systems can simultaneously provide anonymity, privacy and correctness for real-world systems such as electronic voting, anonymous payments, identification systems or private comparisons.

While there has been tremendous progress on improving lattice-based zero-knowledge proofs building (practical) anonymous token systems from quantum-secure assumptions is still an open problem. The only existing works in this direction is the lattice-based oblivious pseudo-random function by Albrecht *et al.* [ADDS21] (mostly a proof of concept) and the blind signature by Lyubashevsky *et al.* [LNS21] (where the running time depends on the number of users in the system). A proposed protocol based on isogenies [BKW20] was later broken [BKM+21].

[1]datatracker.ietf.org/doc/html/rfc8446

[2]signal.org/docs

[3]csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization

Anonymous Communication

During the ongoing Covid-19 pandemic, there has been a strong need to track infected people's movements to stop the spread of the virus. However, doing so in an unencrypted manner can significantly violate the people's privacy and anonymity.

To avoid this, both Google and Apple made available a Bluetooth API such that governments across the world could develop mobile applications that store random identifiers based on which individuals have met. Such an app then uploads these identifiers to a system that send notifications to all close contacts of a person who has tested positive for Covid-19. However, there was a privacy gap in the overall system. Whenever a person tested positive and uploaded the identifier from the app, these identifiers were tied to that user, and could potentially expose that person's locations and social graph.

We, a small team of researchers and software developers, implemented a functionality called anonymous tokens into the Norwegian contact-tracing app, so that all identifiers were uploaded anonymously to the central server. This breaks the link between the user and their stored identifiers. At the height of the pandemic, the app had 1.3 million active users[4]. Afterwards, we introduced the following improvements to the anonymous token protocol.

Paper I *Anonymous Tokens with Public Metadata and Applications to Private Contact Tracing.* We developed the first anonymous token protocol with public metadata, such as expiration dates, built into the tokens. This allows for mass revocation of unspent tokens and solves the challenging problem of frequently rotating the signing and verification keys across several servers. Furthermore, we developed the first publicly verifiable anonymous tokens, allowing for outsourced delegation of either the token signing or verification process, solving the problem of splitting the secret signing key across several servers in the first place. We show that our tokens improve upon the state-of-the-art in natural settings by up to 90% in communication. The security of these protocols is based on discrete logarithm assumptions.

[4]fhi.no/om/smittestopp/nokkeltall-fra-smittestopp

Anonymous tokens are additionally deployed on the Internet for anonymous browsing, spam protection, fraud protection, private file storage, private click measurements, and more. The Internet Engineering Task Force (IETF) is currently standardizing such protocols[5], including the extensions described above. Future research directions involve instantiating this framework based on quantum-secure assumptions and extend this to password-authenticated key-exchange.

Verifiable Shuffles

In some settings we need to break the connection between identities and ciphertexts. We use verifiable shuffles to achieve this. Common use-cases are to ensure privacy of votes in electronic voting or provide anonymity of messages on a communication platform. Here, a server receives a vector of ciphertexts as input, re-randomizes the ciphertexts using homomorphic encryption (by adding randomness to them so that they look different, but decrypt to the same values as before), and permutes the order of the ciphertexts in the vector. The output is a new vector of ciphertexts that decrypt to the same set of messages, but in a different order. If the server does not reveal the randomness nor the permutation to anyone else, this breaks the link between the input and the output to the server. To make sure that this is done honestly, the server must provide a proof of correct computation.

The building blocks described above are often components of a larger system. In such systems, the trust is usually distributed among several parties, assuming that none, one or several of those parties can be trusted to achieve certain properties. A classic example is electronic voting. Here, a ballot box may receive all the encrypted votes from the voters. When the voting period is over, the set of ciphertexts is sent to a series of mixing nodes where each node performs a verifiable shuffle. Finally, the set of ciphertexts is verifiably decrypted. To achieve privacy of the votes, we need at least one mixing server to be honest to break the link between the input ciphertexts and the decrypted votes. If there is more than one decryption server, we need at least one server to be honest and not share their decryption key

[5]datatracker.ietf.org/wg/privacypass

with anyone else. For integrity of the system, we need all mix-nodes and decryption nodes to each provide a proof of correct computation.

Paper II *Lattice-Based Proof of Shuffle and Applications to Electronic Voting.* We presented the first practical post-quantum verifiable shuffle protocol from lattice-based cryptography, showing that this can be computed efficiently in practice. Furthermore, we combined our protocol with a verifiable encryption scheme to design an electronic voting system similar to the system used by Norway for local and national elections in 2011 and 2013. We implemented our system and showed that it was more efficient than any other post-quantum secure voting system in the literature.

Paper III *Verifiable Mix-Nets and Distributed Decryption for Voting from Lattice-Based Assumptions.* We provided the first complete post-quantum secure electronic voting system using lattice-based cryptography. We extended the shuffle from the previous paper into a full mix-node, gave a verifiable distributed decryption protocol, and combined these into a full voting system. We also gave concrete parameters for a system with four mix-nodes and decryption servers, and provided an implementation of each component in the system.

We remark that the above protocols can also be used for single-server verifiable decryption; however, we can get more efficient protocols by tailoring new proof systems for this specific purpose.

Verifiable Decryption

Verifiable decryption is used to prove that one or more ciphertexts decrypt to given messages, without exposing the secret key. This is essential for decrypting messages in systems for anonymous communication or electronic voting (in both cases an adversary could break privacy if they had access to the decryption key and could decrypt messages before they are mixed). It also has applications to homomorphic encryption: for example, a weak client (e.g., a phone) could outsource computation to two untrusted and non-colluding entities, where one entity handles encryption and decryption while the other entity handles ciphertext computation. This way, the client can outsource all cryptographic operations. Such applications usually require the decryption of a large number of ciphertexts. We provide two efficient and simple protocols for efficiently and verifiably decrypting large sets of ciphertexts.

Paper IV *Verifiable Decryption in the Head.* We designed a verifiable decryption protocol using cut-and-choose techniques instead of heavy zero-knowledge proofs. The protocol is very lightweight in computation and communication, but must be run many times to reduce the probability of cheating. The setup is very cheap when amortizing over many ciphertexts, and we provide an efficient and simple verifiable decryption protocol per ciphertext compared to the state-of-the-art. We also provide a proof-of-concept implementation to showcase its practicality.

Paper V *Short paper: Verifiable Decryption for BGV.* We designed a direct construction for verifiable decryption using newly developed zero-knowledge protocols from the literature. This protocol does not require a heavy setup, and the cost of decryption itself is very low when working on batches of a few thousand ciphertexts at once. We also provide a proof-of-concept implementation to showcase its practicality.

Implementing cryptographic protocols is hard [HLPT20], and we have seen many attacks both in theory and practice. Both schemes above are conceptually simpler and easier to understand than the previous state-of-the-art, making them more straightforward to implement than previous works [LNS21]. They are comparable or better in communication cost when amortizing over many ciphertexts. Our proof-of-concept implementations prove their efficiency, and our design makes them very attractive for practical use.

References

[AASA$^+$20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the second round of the nist post-quantum cryptography standardization process, 2020. `https://doi.org/10.6028/NIST.IR.8309`.

[ADDS21] Martin R. Albrecht, Alex Davidson, Amit Deo, and Nigel P. Smart. Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 261–289. Springer, Heidelberg, May 2021.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008*, pages 335–348. USENIX Association, July / August 2008.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems. *Electron. Colloquium Comput. Complex.*, 3, 1996.

[ALS20] Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler. Practical product proofs for lattice commitments. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 470–499. Springer, Heidelberg, August 2020.

[BBC$^+$18] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sublinear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 669–699. Springer, Heidelberg, August 2018.

[BDH11] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 117–129. Springer, Heidelberg, November / December 2011.

[BDL+18] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 368–385. Springer, Heidelberg, September 2018.

[Beu20] Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 183–211. Springer, Heidelberg, May 2020.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

[BHH+15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: Practical stateless hash-based signatures. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 368–397. Springer, Heidelberg, April 2015.

[BKLP15] Fabrice Benhamouda, Stephan Krenn, Vadim Lyubashevsky, and Krzysztof Pietrzak. Efficient zero-knowledge proofs for commitments from learning with errors over rings. In Günther Pernul, Peter Y. A. Ryan, and Edgar R.

Weippl, editors, *ESORICS 2015, Part I*, volume 9326 of *LNCS*, pages 305–325. Springer, Heidelberg, September 2015.

[BKM$^+$21] Andrea Basso, Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Antonio Sanso. Cryptanalysis of an oblivious prf from supersingular isogenies. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 160–184, Cham, 2021. Springer International Publishing.

[BKW20] Dan Boneh, Dmitry Kogan, and Katharine Woo. Oblivious pseudorandom functions from isogenies. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 520–550. Springer, Heidelberg, December 2020.

[BLS19] Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler. Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 176–202. Springer, Heidelberg, August 2019.

[CWF13] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In Samuel T. King, editor, *USENIX Security 2013*, pages 147–162. USENIX Association, August 2013.

[DGS$^+$18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018.

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

[ENS20] Muhammed F. Esgin, Ngoc Khanh Nguyen, and Gregor Seiler. Practical exact proofs from lattices: New techniques to exploit fully-splitting rings. Cryptology ePrint Archive, Report 2020/518, 2020. `https://eprint.iacr.org/2020/518`.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

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

[Gjø11] Kristian Gjøsteen. The norwegian internet voting protocol. In *E-Voting and Identity - Third International Conference, VoteID 2011*, pages 1–18, 2011.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[HIJ$^+$21] Sharon Huang, Subodh Iyengar, Sundar Jeyaraman, Shiv Kushwah, Chen-Kuei Lee, Zutian Luo, Payman Mohassel, Ananth Raghunathan, Shaahid Shaikh, Yen-Chieh Sung, and Albert Zhang. Dit: De-identified authenticated telemetry at scale. Technical report, Facebook Inc., `https://research.fb.com/wp-content/uploads/2021/04/DIT-De-Identified-Authenticated-Telemetry-at-Scale_final.pdf`, 2021.

[HLPT20] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy*, pages 644–660. IEEE Computer Society Press, May 2020.

[JD11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34. Springer, Heidelberg, November / December 2011.

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

[LNS20] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Practical lattice-based zero-knowledge proofs for integer relations. CCS 2020, 2020.

[LNS21] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 215–241. Springer, Heidelberg, May 2021.

[LPT19] Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. Trapdoor commitments in the SwissPost e-voting shuffle proof, 2019.

[LW18] Fucai Luo and Kunpeng Wang. Verifiable decryption for fully homomorphic encryption. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *ISC 2018*, volume 11060 of *LNCS*, pages 347–365. Springer, Heidelberg, September 2018.

[McE78] R. McEliece. A public key cryptosystem based on algebraic coding theory. 1978.

[MI88] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynominal-tuples for efficient signature-verification and message-encryption. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 419–453. Springer, Heidelberg, May 1988.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer, Heidelberg, August 1986.

[Nie86] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. 1986.

[Pat95] Jacques Patarin. Cryptanalysis of the Matsumoto and Imai public key scheme of eurocrypt'88. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 248–261. Springer, Heidelberg, August 1995.

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

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

[Sho97] P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26:1484–1509, 1997.

[Sto10] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves, 2010.

[T$^+$20] Carmela Troncoso et al. Decentralized privacy-preserving proximity tracing. `https://arxiv.org/abs/2005.12273`, 2020.

[YAZ$^+$19] Rupeng Yang, Man Ho Au, Zhenfei Zhang, Qiuliang Xu, Zuoxia Yu, and William Whyte. Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 147–175. Springer, Heidelberg, August 2019.

Paper I

Anonymous Tokens with Public Metadata and Applications to Private Contact Tracing

Tjerand Silde and Martin Strand

Accepted at Financial Cryptography and Data Security, FC 2022. The full version is available at: eprint.iacr.org/2021/203.pdf.

Anonymous Tokens with Public Metadata and Applications to Private Contact Tracing

Tjerand Silde[1] and Martin Strand[2]

[1] Department of Mathematical Sciences
Norwegian University of Science and Technology – NTNU
`tjerand.silde@ntnu.no`
[2] Norwegian Defence Research Establishment – FFI
`martin.strand@ffi.no`

Abstract. Anonymous single-use tokens have seen recent applications in private Internet browsing and anonymous statistics collection. We develop new schemes in order to include public metadata such as expiration dates for tokens. This inclusion enables planned mass revocation of tokens without distributing new keys, which for natural instantiations can give 77 % and 90 % amortized traffic savings compared to Privacy Pass (Davidson *et al.*, 2018) and DIT: De-Identified Authenticated Telemetry at Scale (Huang *et al.*, 2021), respectively. By transforming the public key, we are able to append public metadata to several existing protocols essentially without increasing computation or communication.
Additional contributions include expanded definitions, a more complete framework for anonymous single-use tokens and a description of how anonymous tokens can improve the privacy in dp^{3}t-like digital contact tracing applications. We also extend the protocol to create efficient and conceptually simple tokens with both public and private metadata, and tokens with public metadata and public verifiability from pairings.

Keywords: anonymous tokens · public metadata · contact tracing

1 Introduction

Anonymous credentials have been an active research area since the 1980's [Cha82, Cha83], involving schemes such as blind signatures, partially blind signatures, anonymous tokens, attribute-based credentials, group signatures, ring signatures etc. This enables more complex systems for e.g., electronic cash or electronic voting, but also, to protect the privacy of the users in chat applications like Signal.

Recent work by Davidson *et al.* [DGS$^+$18] presents a very practical protocol, named Privacy Pass [DGS$^+$], for anonymous single-use tokens. This protocol allows users to browse anonymously, e.g., using Tor, without having to solve a CAPTCHA every time they visit a website. Privacy Pass gives the user a set of randomized tokens whenever they solve a CAPTCHA, which they then later can redeem instead of solving a new CAPTCHA. This improves the usability of anonymous browsing. It also gives protection against spam, prevents DDoS attacks and provides fraud resistance without the need for cross-site tracking or fingerprinting. However, the only way to expire or revoke batches of unspent tokens is by replacing the private-public key pair in a trusted way, which is impractical [Dav21].

Privacy Pass has gained a lot of attention, and is currently being integrated to improve privacy in several applications, e.g., for private file storage[3] and for basic attention tokens (BATs) in the Brave browser[4]. It can also be used for private click measurement when making a purchase or signing up for a service[5].

Facebook uses partially blind signatures for combating fraud [IT21], and they have developed an extension of Privacy Pass called DIT: De-Identified Authenticated Telemetry at Scale [HIJ$^+$21], which is used for privately collecting client-side telemetry from WhatsApp. DIT requires daily key-rotation to prevent DoS attacks, which led to the development of an attribute-based verifiable oblivious pseudorandom function for transparent key-rotation.

The IETF is currently standardizing Privacy Pass [Int21], while Trust Token [Wor21] is currently being standardized by the World Wide Web Consortium. Both standardization processes mention private and public metadata, in addition to public verifiability, as desirable extensions to the Privacy Pass protocol. Public metadata allows for more efficient key-rotation, and opens for applications using public labeling and public anonymity sets, while private metadata allows for allow/deny lists, rate-limiting, or trust-indication. Public verifiability allows for outsourcing signing or verification of tokens.

Kreuter *et al.* [KLOR20a] gave the first construction of anonymous tokens with private metadata, while we give the first construction with public metadata. Our construction can also be combined with private metadata or public verifiability.

Privacy Pass guarantees anonymity for all tokens generated by the same key. The addition of any metadata reduces the anonymity set. We have designed the protocol in such a way that the user and the signer must agree on the metadata. Any application should restrict its use of metadata to generic, predefined values that would otherwise have triggered a change of keys, e.g., expiry dates. Client software should validate that the metadata is in accordance to the policy, and reject any malformed tokens. Furthermore, private metadata bits also reduces the anonymity set. Our protocol can easily be extended to include more than one private metadata bit, but this must be done with great care, as it opens for secretly tracking smaller sets of individual users.

Independently of this work, Tyagi *et al.* [TCR$^+$21] have proposed a similar construction to include public metadata, along with a novel hardness assumption and a reduction to a more conventional problem, to be used in partially oblivious pseudo-random functions. We discuss their work further in Sections 1.5 and 4.

[3] PrivateStorage: medium.com/least-authority/the-path-from-s4-to-privatestorage-ae9d4a10b2ae.

[4] Brave: github.com/brave/brave-browser/wiki/Security-and-privacy-model-for-ad-confirmations.

[5] Private Click Measurement: privacycg.github.io/private-click-measurement.

1.1 Background

Adopting the terminology from Privacy Pass [DGS$^+$18], we have the following informal architecture. A *user* asks the *signer* for a one-time anonymous token in a *signing phase*, later to be redeemed to a *verifier* in a *redemption phase*.

The literature provides many flavors of anonymous credentials. They all come with some minimal requirements with respect to security, and have a variation of desirable properties for practical applications.

Unlinkability and Unforgeability. To ensure privacy of users, the anonymous token protocol must make sure that the information being transferred in the attestation phase cannot be correlated with the token being received in the redemption phase. This is called unlinkability. To ensure the system's integrity, the anonymous token protocol must make sure that users cannot forge tokens, even after receiving valid tokens from the attestation server. This is called unforgeability. These are the minimal requirements for anonymity and integrity, and have been handled starting from the first classical constructions of blind signatures starting with David Chaum in the early 1980's [Cha82, Cha83] and subsequent work on anonymous credentials [Oka93, PS96, CL01, CL03, CL04, CHL05, CG08, GMS10, BL13, CMZ14, CPZ20].

Underlying Primitives. Anonymous token protocols can be built from a variety of cryptographic primitives and assumptions, e.g., factoring [AF96, CL01, CL03], discrete logarithms [Oka93, AO00, WSMZ06, DGS$^+$18, KLOR20a] or bilinear pairings [ZSS03, CHYC05, CZMS06, BPV12]. Protocols based on elliptic curves are the most efficient, both in terms of size and timings, while other primitives might more easily provide correctness and verifiability.

Verifiability and Key-Sharing. In situations where the same party is both attesting and redeeming tokens [DGS$^+$18, KLOR20a], it is natural for the server(s) to share a key. In the designated verifier setting, it is necessary for the attestation server to provide zero-knowledge proofs to ensure that a token is honestly generated [DGS$^+$18, KLOR20a], while pairings can provide public verifiability directly [ZSS03].

Key-Rotation and Token-Revocation. To avoid misuse, it is important to have a mechanism to efficiently expire or revoke batches of tokens. This may be useful for rate limiting to avoid denial-of-service attacks, or to protect users from credential stuffing [DGS$^+$18, TPY$^+$19, HIJ$^+$21]. In Privacy Pass [DGS$^+$18], this is solved by infrequent key-rotation where a few public keys are available at a public endpoint. DIT [HIJ$^+$21], which rotates their keys every day, solves it by using a new attribute-based verifiable oblivious pseudorandom function (VOPRF). However, both solutions are inconvenient in practice, and add significant overhead for validating the public keys.

Rounds of Interaction. Blind signatures and anonymous tokens can be attested in only one round of communication, which is optimal. This saves time and computation for both the client and the server, and the parties does not need to keep a state. However, the only partially blind signatures achieving one round of interaction are based on bilinear pairings or factoring [AF96], while protocols based on discrete logarithms [AO00, WSMZ06] needs two rounds. We note that there is no one-round single-use anonymous token protocol with efficient revocation in the literature with security based on elliptic curve discrete logarithms without pairings before this work. Popular schemes such as [CL04,BL13] require at least two rounds of communication.

1.2 Our Contribution

Our contribution in this paper is threefold: first, we present new definitions and a new framework for anonymous tokens – extending the work by Kreuter *et al.* [KLOR20a] – to also consider public metadata and/or public verifiability. Secondly, we present three efficient protocols for anonymous tokens with efficient batched revocation: 1) Privacy Pass [DGS$^+$18] with public metadata, 2) Kreuter *et al.* [KLOR20a] with public and private metadata, and 3) a Privacy Pass inspired protocol using pairings to satisfy public verifiability while including public metadata. Thirdly, we present contact tracing as a new and important application for anonymous single-use tokens, and discuss the implementation of Privacy Pass used in the Norwegian contact tracing app Smittestopp to improve users' privacy.

Updated Definitions and New Framework. Several works have asked for efficient batched revocation of anonymous tokens without key-rotation [DGS$^+$18, Dav21]. Additionally, there is a need for anonymous tokens with public verifiability [Wor21], so that token generation can be delegated, and verification can be performed locally for token redemption. We provide updated definitions for all of these cases: designated verifier anonymous tokens with or without public and/or private metadata and public verifier anonymous tokens with and without public metadata. Details can be found in Section 3.

Anonymous Tokens with Public Metadata. We present the first anonymous tokens protocols with efficient batched revocation, meaning that the protocol only requires one round of communication based on lightweight primitives and that we avoid key-rotation. The key insight in our protocol is conceptually very simple: all parties locally update the public key based on the hash of the public metadata, and then execute the protocols with respect to the new key pair. The main challenge is to sign tokens in a way that does not allow the user to forge tokens initially signed under metadata md to be valid under metadata md' instead. Let k be the secret key and let $d = \mathtt{H}(\mathsf{md})$ be the hash of the metadata. Our solution, inspired by Zhang *et al.* [ZSS03], is to use the inverse $e = (d+k)^{-1}$

as the new signing key. This allows us to replace the secret keys in the previous protocols in a modular way.

Furthermore, to avoid subliminal channels, the signer needs to prove that the signed token is computed correctly. This is easily solved for Privacy Pass [DGS+18]. In the original protocol they use a zero-knowledge protocol to prove, given generator G, public key $K = [k]G$, blinded token T' and signed token $W' = [k]T'$, the equality of discrete logarithms $\log_G K = k = \log_{T'} W'$ to ensure correctness. In our updated protocol, including metadata md, updated public key $U = [d]G + K$ and signed token $W' = [e]T'$, we prove the equality of discrete logarithms $\log_G U = d + k = \log_{W'} T'$ to ensure correctness.

However, it is not as easy to ensure correctness in the extended version of the protocol by Kreuter *et al.* [KLOR20a] including both public and private metadata. We solve this by combining an `OR`-proof with two `AND`-proofs to make sure that the correct key is used. Further improvement is an open problem.

Next, we give a protocol based on pairings. The protocol is an adapted version of the partially blind signatures by Zhang *et al.* [ZSS03], where we tweak it into the same structure as Privacy Pass. We note that the communication in the protocol is the same, but in addition to get a more streamlined protocol structure, we also allow for more efficient instantiation in practice using the BLS12-381 pairing [BLS03]. Ideally, we would like to avoid pairings altogether, but this seems necessary in practice. See more details about the protocols in Section 4.

Finally, we detail the communication efficiency of the protocols in Section 5, and compare our constructions with the current state of the art with respect to efficient batched revocation in Table 1. We show that our protocols are much more efficient in practice. We also make a concrete comparison with DIT [HIJ+21] for collecting telemetry-data from WhatsApp, and show that our protocol in Figure 6 would decrease the size of the signed token in a natural setting by 90 %, saving the Facebook servers up to 1.7 TB of communication every day.

More Private Contact Tracing. Many countries have recently developed contact tracing apps as one of the measurements to battle the ongoing pandemic. These apps are inherently storing sensitive information about the user, e.g., the users' location graph and social graph. To avoid large, centralized databases with such sensitive information about a large portion of a country's adult population, most apps are based on the decentralized Google/Apple Exposure Notification System (ENS). However, there are still privacy issues with regards to uploading the randomized exposure keys to the central server, as the user would have to identify themselves to ensure that only people who have tested positive for COVID-19 are able to upload keys. We implemented Privacy Pass into the Norwegian contact tracing app to improve the users' privacy. Our code is published at github.com/HenrikWM/anonymous-tokens, and the Norwegian Institute of Public Health (NIPH) have made the source code for the contact tracing infrastructure publicly available[6]. We present more details about the contact tracing infrastructure and improvements in Section 6.

1.3 Comparison to Anonymous Credentials

There is a long line of research on more generalized anonymous credentials with features such as multi-show, multi-attributes, and revocability – in addition to the mandatory unlinkability and unforgeability – that allow one to encode expiration dates as attributes.

However, generalized anonymous credentials often depends on stronger assumptions, e.g., strong RSA [CL01, CV02, CL03, CHL05, CG08], strong Diffie-Hellman [AMO08] or DL assumptions in bilinear groups [CL04, HS21]. Some schemes only depend on DDH [BL13, PZ13, CMZ14, CPZ20], but these schemes require larger messages in general. In conclusion, generalized anonymous credentials inherently impose larger parameters, more rounds of communication and less efficient protocols in practice, resulting in thousands of bits on communication over multiple rounds.

Finally, more general and complex anonymous credentials make these schemes less suited for use in simpler single-use systems with many users, which is the case in our setting. We want to minimize the rounds of communication and data being sent, in addition to minimizing the local computation and the local state. Hence, we only compare to one-round single-use efficiently revocable anonymous credentials with minimal communication in Section 5.

1.4 Related work

Our work achieving designated verification and public metadata extends a long line of publications. Freedman *et al.* [FIPR05] introduced oblivious pseudorandom functions, and Jarecki *et al.* [JKK14, JKX18] gave an efficient instantiation based on DDH in the random oracle model. Papadopoulos *et al.* [PWH$^+$17] gave a verifiable PRF from elliptic curves, and Burns *et al.* [BMR$^+$17] gave an oblivious PRF from elliptic curves. Privacy Pass combined these results with an extended version of the Chaum-Pedersen zero-knowledge protocol [CP93] given by Henry and Goldberg [HG13, Hen14] to prove knowledge of batches of elements having the same discrete logarithm, and Kreuter *et al.* [KLOR20a] added private metadata to Privacy Pass. In a concurrent work, Tyagi *et al.* [TCR$^+$21] recently extended this line of works to partially oblivious PRFs.

To achieve public verifiability we use parings, inspired by the seminal work of Boneh *et al.* [BLS01] for short and efficient signatures and a series of constructions of (partially) blind signatures based on pairings [ZSS03, Bol03, CHYC05, CZMS06, CKS09, BPV12, FHS15, FHKS16].

[6] NIPH: github.com/folkehelseinstituttet/?q=Smittestopp.

1.5 Chronology

As we report on both an implementation and new protocols, we believe it can be helpful to lay out the chronology of this work to separate the contributions.

Mid-October 2020, the authors were made aware of a potential privacy weakness in Norway's upcoming second COVID-19 contact tracing app Smittestopp. The first iteration had been stopped by the Norwegian Data Protection Agency in June, due to privacy concerns following from lack of data minimization. The new app had a set launch date in December.

The issue was that the verification service would collect IDs in order to automatically verify the infection status, and then send a token to the app which could then be used for uploading exposure keys. This token would create a hard link between an ID-based service and the rest of the system, in which the users are assumed to be anonymous.

Within a few days, we suggested using Privacy Pass in order to remove this link. Due to lack of capacity, our proposal was acknowledged, but we were asked to provide the code. We teamed up with Henrik Walker Moe to implement Privacy Pass in C#, and our implementation was eventually accepted into Smittestopp along with an improvised solution to rotate keys every three days.

Motivated by this process and the last-minute improvisation, we expanded the original Privacy Pass protocol to deal with the issues of key-rotation and revocation. Our initial manuscript was posted on ePrint February 24th, 2021. We were then made aware of a complication to the security proof, which was originally from the work by Zhang *et al.* [ZSS03]. A correct proof was posted on ePrint by Tyagi *et al.* [TCR$^+$21] June 24th, 2021. The primary separation between these two manuscripts are that we were the first to present this protocol along with its variations, while Tyagi *et al.* present a correct proof. We also present the protocols in a way that is compatible to previous work. In this sense, these works complement each other.

The new protocol has not been implemented in Smittestopp. This is due to lack of further development of the app, and we do not expect any major changes to be accepted into the codebase at this stage.

2 Preliminaries

We assume that the reader is familiar with the basics of elliptic curve cryptography. To fix notation, let q be a prime and let $\mathbb{F}_{q^\ell}$ for some $\ell > 0$ be a field of characteristic q. Let E be all points (x, y) that satisfy the elliptic curve equation $y^2 = x^3 + ax + b$ in the algebraic closure of $\mathbb{F}_{q^\ell}$, and let $E(\mathbb{F}_{q^\ell})$ denote the set of all such points in $\mathbb{F}_{q^\ell} \times \mathbb{F}_{q^\ell}$ along with the point at infinity $\mathcal{O}$. By abuse of notation, we often let E be a group of order p inside $E(\mathbb{F}_{q^\ell})$. Define the group law in the usual additive way. In particular, let $[m] : E \to E$ be the multiplication-by-m map, which takes the same role as exponentiation in multiplicative groups. Now follows a brief discussion of the Chosen-Target Gap Diffie-Hellman problem and some zero-knowledge proofs we will need as primitives.

2.1 The One-More Gap Strong Diffie-Hellman Problem

The strong Diffie-Hellman problem was introduced by Boneh and Boyen [BB04]. Given a sequence $g, g^x, g^{x^2}, \dots, g^{x^q}$ from a group $\mathbb{G}$ of prime order p, output a pair $(c, g^{(x+c)^{-1}})$ with $c \in \mathbb{Z}_p$. We now present a variant of this game: the adversary must commit to fixed set of candidates $\{c_i\}$, and may then query an oracle for $B^{(\mathsf{sk}+c_i)^{-1}}$ for arbitrary B, along with an oracle for the decision variant. The adversary wins if it can present $\ell + 1$ correct tuples for a chosen c_i, but only having queried ℓ or less. The details are presented in Figure 1. The definition and game is due to Tyagi *et al.* [TCR+21].

```
Game $(m,n)$-OM-GAP-SDHI$_{\mathsf{Gen},\mathcal{A},\ell}(\lambda)$
$(\mathbb{G}, p, g) \leftarrow \mathsf{Gen}(1^\lambda)$
$\mathsf{sk} \leftarrow\$ \mathbb{Z}_p$
$(y_i)_{i\in[m]} \leftarrow\$ \mathbb{Z}_p^m$
$(\mathsf{st}_\mathcal{A}, (c_i)_{i\in[n]}) \leftarrow \mathcal{A}_1(p, \mathbb{G})$
$\left(\gamma, (Z_i, \alpha_i)_{i\in[\ell+1]}\right) \leftarrow \mathcal{A}_2^{\mathrm{SDH,SDDH}}\left(g, g^{\mathsf{sk}}, (g^{y_i})_{i\in[m]}; \mathsf{st}_\mathcal{A}\right)$
if $q_\gamma \le \ell$ and $(i \ne j \to \alpha_i \ne \alpha_j)$ then
    return $(Z_i)_{i\in[\ell+1]} = \left(g^{y_{\alpha_i}(\mathsf{sk}+c_\gamma)^{-1}}\right)_{i\in[\ell+1]}$

Oracle SDH$(B, i)$
if $i \notin [n]$ then
    return $\perp$
$q_i := q_i + 1$
$Z := B^{(\mathsf{sk}+c_i)^{-1}}$
return $Z$

Oracle SDDH$(Y, Z, i)$
return $Z = Y^{(\mathsf{sk}+c_i)^{-1}}$
```

Fig. 1. The one-more gap strong inversion Diffie-Hellman security game.

Definition 1 **((m,n)-One-More Gap Strong Inversion Diffie-Hellman).** *Let m, n be natural numbers, and let $\mathbb{G}$ be a cyclic group of order p with generator g produced by the algorithm $\mathsf{Gen}(1^\lambda)$. Let (m,n)-*OM-GAP-SDHI* be the game defined in Figure 1. (m,n)-One-More Gap Strong Diffie-Hellman Inversion holds for $\mathbb{G}$ if for any* PPT *adversary $\mathcal{A}$ and any $\ell \ge 0$,*

$$\mathsf{Adv}^{\text{om-gap-sdhi}}_{\mathsf{Gen},\mathcal{A},\ell}(\lambda) := \Pr[(m,n)\text{-OM-GAP-SDHI}_{\mathsf{Gen},\mathcal{A},\ell}(\lambda) = 1] = \mathsf{negl}(\lambda).$$

Tyagi *et al.* [TCR+21] have proven that this assumption is implied by the much simpler q-DL assumption, which asks the adversary to return x, given $g, g^x, g^{x^2}, \dots, g^{x^q}$.

2.2 DDH vs. CDH in Pairings

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order, written additively, and let $\mathbb{G}_T$ be another cyclic group of same prime order, written multiplicatively. A bilinear pairing $\hat{e}$ is a map

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

such that the following properties hold:

Bilinearity For all $P_1, P_2 \in \mathbb{G}_1$ and $Q_1, Q_2 \in \mathbb{G}_2$, it holds that $\hat{e}(P_1+P_2, Q_1) = \hat{e}(P_1, Q_1)\hat{e}(P_2, Q_1)$ and $\hat{e}(P_1, Q_1 + Q_2) = \hat{e}(P_1, Q_1)e(P_1, Q_2)$.
Non-degeneracy For all $P \neq \mathcal{O}$, $\hat{e}(P, P) \neq 1$.
Computability $\hat{e}$ can be efficiently computed.

The bilinearity property implies that for scalars a, b, we have $\hat{e}([a]P, [b]Q) = \hat{e}(P, Q)^{ab}$, which is the crucial property used for verification.

Bilinear maps lend themselves to a variant of the well-known Diffie-Hellman problem, the Chosen-Target Gap Diffie-Hellman problem [BNPS02]. Even if the adversary is given oracle access to ℓ instances of the Computational Diffie-Hellman (CDH) problem and arbitrary many queries to a Decision Diffie-Hellman (DDH) oracle, it should still not be able to compute the final Diffie-Hellman instance $\ell + 1$. We repeat the game and definition by Kreuter *et al.* [KLOR20a].

```
Game $\mathrm{CTGDH}_{\mathsf{Gen},\mathcal{A},\ell}(\lambda)$
$\Gamma = (\mathbb{G}, p, G) \leftarrow \mathsf{Gen}(1^\lambda)$
$x \leftarrow\!\!\$\ \mathbb{Z}_p; X := [x]G$
$q := 0; \mathcal{Q} := [\,]$
$(t_i, Z_i)_{i\in[\ell+1]} \leftarrow \mathcal{A}^{\textsc{Target},\textsc{Help},\mathrm{DDH}}(\Gamma, X)$
for $i \in [\ell + 1]$
    if $t_i \notin \mathcal{Q}$ then return $0$
    $Y_i := \mathcal{Q}[t_i]$
return $(q \leq \ell$ and
    $\forall i \neq j \in [\ell + 1], t_i \neq t_j$ and
    $\forall i \in [\ell + 1], [x]Y_i = Z_i)$
```

```
Oracle $\textsc{Target}(t)$
if $t \in \mathcal{Q}$ then
    $Y := \mathcal{Q}[t]$
else
    $Y \leftarrow\!\!\$\ \mathbb{G}$
    $\mathcal{Q}[t] := Y$
return $Y$
```

```
Oracle $\textsc{Help}(Y)$
$q := q + 1$
return $[x]Y$
```

```
Oracle $\mathrm{DDH}(Y, Z)$
return $(Z = [x]Y)$
```

Fig. 2. The Chosen-target gap Diffie-Hellman security game.

Definition 2 (Chosen-Target Gap Diffie-Hellman). *Let $\mathbb{G}$ be a cyclic group of order p with generator G produced by the algorithm $\mathsf{Gen}(1^\lambda)$. Let* CTGDH *be the game defined in Figure 2. Chosen-Target Gap Diffie-Hellman holds for $\mathbb{G}$ if for any* PPT *adversary $\mathcal{A}$ and any $\ell \geq 0$,*

$$\mathsf{Adv}^{\mathrm{ctgdh}}_{\mathsf{Gen},\mathcal{A},\ell}(\lambda) := \Pr[\mathrm{CTGDH}_{\mathsf{Gen},\mathcal{A},\ell}(\lambda) = 1] = \mathsf{negl}(\lambda).$$

2.3 Proof of Equal Discrete Logs

Chaum and Pedersen [CP93] introduced an elegant honest-verifier zero-knowledge protocol to prove that two group elements have the same discrete logarithm relative to their respective bases, $\log_G K = k = \log_T W$. We describe the protocol loosely to ensure the reader is familiar with the idea. Let $\mathbb{G}$ be a cyclic group of prime order p with independent generators G and T, and let $K := [k]G$, $W := [k]T$ where k is a scalar private to the prover $\mathcal{P}$.

$\mathcal{P}$.1 Choose a random scalar r in the underlying field, compute $A := [r]G$ and $B := [r]T$, and send (A, B) to $\mathcal{V}$.
$\mathcal{V}$.1 Choose a random challenge c modulo p and send it to $\mathcal{P}$.
$\mathcal{P}$.2 Compute the response $z := r - ck$ modulo p, and then send z to $\mathcal{V}$.
$\mathcal{V}$.2 Verify that $A = [z]G + [c]K$ and $B = [z]T + [c]W$.

This protocol satisfies unconditional special soundness and special honest-verifier zero-knowledge. One can make the protocol non-interactive by applying the Fiat-Shamir transformation [FS87]. The prover queries the oracle on the tuple $(\mathbb{G}, G, T, K, W, A, B)$. In addition, one can reduce communication by sending the oracle response c instead of (A, B), and modifying the final verification step to querying the oracle on $(\mathbb{G}, G, T, K, W, [z]G + [c]K, [z]T + [c]W)$, and then verify that it indeed returns c. We will use a shorthand notation to refer to this proof as $\Pi_{\text{DLEQ}}(G, T, K, W; k)$, meaning that $\log_G([k]G) = \log_W T$.

The proof can be batched for many instances with respect to the same secret scalar using the techniques by Henry [Hen14] as showed in [DGS+18, Section 3.2.1].

2.4 AND-Proof of Equal Discrete Logs

Let $\mathbb{G}$ be an additive group of prime order p with independent generators G, H, T, S, and let $K := [k_0]G + [k_1]H$ and $V := [k_0]T + [k_1]S$, where k_0, k_1 are scalars private to the prover $\mathcal{P}$. We want to prove that V is correctly computed with respect to T and S using the same secret scalars as K with respect to G and H. We present a simple protocol to prove this relation, by essentially computing two Chaum-Pedersen proofs in parallel.

$\mathcal{P}$.1 Choose two random scalars r_0, r_1 modulo p. Compute $A := [r_0]G + [r_1]H$, $B := [r_0]T + [r_1]S$, and send (A, B) to $\mathcal{V}$.
$\mathcal{V}$.1 Choose a random challenge c modulo p and send it to $\mathcal{P}$.
$\mathcal{P}$.2 Compute $z_0 := r_0 - ck_0$ and $z_1 := r_1 - ck_1$ modulo p and send them to $\mathcal{V}$.
$\mathcal{V}$.2 Verify that $A = [c]K + [z_0]G + [z_1]H$ and $B = [c]V + [z_0]T + [z_1]S$.

It is straightforward to verify that this is a sigma protocol with special soundness and special honest-verifier zero-knowledge. As above, we can apply the Fiat-Shamir [FS87] transformation to get a non-interactive protocol. We will refer to this proof as $\Pi_{\text{DLEQ2}}(G, H, T, S, K, V; k_0, k_1)$.

2.5 OR-Proof of Equal Discrete Logs

We present the honest-verifier zero-knowledge `OR`-proof of equal discrete logarithms instantiated by Kreuter et al. [KLOR20b, Appendix B]. Let $\mathbb{G}$ be a cyclic group of prime order p with generators G, H, T, S, and let $V_0 := [e_{0,0}]G + [e_{0,1}]H$ and $V_1 := [e_{1,0}]G + [e_{1,1}]H$, where $e_{i,j}$ are distinct scalars private to the prover $\mathcal{P}$. Furthermore, let $W := [e_{b,0}]T + [e_{b,1}]S$ for $b \in \{0, 1\}$. We want to prove that W is computed using the same secret scalars as either V_0 or V_1.

$\mathcal{P}$.1 Choose random scalars $r_0, r_1, c_{b-1}, u_{b-1}, v_{b-1}$ in the underlying field and compute the following:

$$\begin{aligned} A_{b,0} &:= [r_0]G + [r_1]H, \\ A_{b,1} &:= [r_0]T + [r_1]S, \\ A_{1-b,0} &:= [u_{b-1}]G + [v_{b-1}]H - [c_{b-1}]V_{b-1}, \\ A_{1-b,1} &:= [u_{b-1}]T + [v_{b-1}]S - [c_{b-1}]W. \end{aligned}$$

Finally, send $(A_{0,0}, A_{0,1}A_{1,0}, A_{1,1})$ to $\mathcal{V}$.

$\mathcal{V}$.1 Choose a random challenge c modulo p and send it to $\mathcal{P}$.

$\mathcal{P}$.2 Compute the responses

$$c_b := c - c_{1-b}, \quad u_b := r_0 + c_b e_{b,0}, \quad v_b := r_1 + c_b e_{b,1},$$

modulo p. Send $(c_i, u_i, v_i)_{i=0,1}$ to $\mathcal{V}$.

$\mathcal{V}$.2 Verify that $c = c_0 + c_1$ and that

$$\begin{aligned} A_{0,0} &= [u_0]G + [v_0]H - [c_0]V_0, \\ A_{0,1} &= [u_0]T + [v_0]S - [c_0]W, \\ A_{1,0} &= [u_1]G + [v_1]H - [c_1]V_1, \\ A_{1,1} &= [u_1]T + [v_1]S - [c_1]W. \end{aligned}$$

We can make the proof non-interactive using Fiat-Shamir [FS87] like above. We will refer to this protocol as $\Pi_{\text{DLEQOR2}}(G, H, T, S, V_0, V_1, W; e_{b,0}, e_{b,1})$.

Π_{DLEQOR2} can be batched for many instances with respect to the same secret scalars using the techniques by Henry [Hen14] as shown in [KLOR20a, Appendix B.1].

3 Definitions for Anonymous Tokens

Anonymous tokens as used in Privacy Pass are conceptually simple: both issuance and verification require the private key, and the final token is uniquely determined by the token seed t and the private key. Kreuter *et al.* [KLOR20a] extended this notion by adding a private bit in the token. We further extend the definition in two different directions: we want to add public metadata, and we want to make the token publicly verifiable. Now, private bits do not make immediate sense in the context of a publicly verifiable token scheme, but public metadata can be relevant in both settings.

The metadata can for instance be used to indicate an expiry date, replacing the need for frequent key rotation in certain applications [HIJ$^+$21] as we discussed in Section 1.1. We model it as a value that the user and issuer must agree upon, which should restrict the issuer from using arbitrary, identifiable values.

Lending terminology from programming, we would like the definition to provide backwards compatibility, and handle the notational incompatibility between private and public verifiability. To this end, we imitate the notion of [optional arguments] from programming. The notation $\mathsf{vk}|\mathsf{sk}$ is meant as "at least one of the public or the secret key". We align our definitions as close as possible to those by Kreuter *et al.* [KLOR20a].

Definition 3 (Anonymous tokens). *An anonymous token scheme with zero or more of **private metadata bit**, **public metadata**, or **public verifiability** consists of the following algorithms:*

– $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{AT.Setup}(1^\lambda)$, *the setup algorithm that takes as input the security parameter λ in unary form, and returns a common reference string* crs *and trapdoor* td. *All the remaining algorithms take* crs *as their first input.*

– $(\mathsf{pp}, \mathsf{sk}, [\mathsf{vk}]) \leftarrow \mathsf{AT.KGen}(\mathsf{crs})$, *the key generation algorithm that generates a signing key* sk *and optionally a verification key* vk *along with public parameters* pp. *All the remaining algorithms take* pp *as their second input.*

– $\sigma \leftarrow \langle \mathsf{AT.User}(\mathsf{pp}, [\mathsf{vk}], t, [\mathsf{md}]), \mathsf{AT.Sign}(\mathsf{sk}, [\mathsf{md}], [b])\rangle$, *the token issuance protocol, which involves interactive algorithms* $\mathsf{AT.User}$ *and* $\mathsf{AT.Sign}$. *The user algorithm takes as input values the public parameters and the token seed $t \in \{0,1\}^\lambda$, and potentially the verification key* vk *and the public metadata* md. *The signing algorithm takes the private key* sk *and potentially metadata* md *and the private bit b. At the end of the interaction, the issuer outputs nothing, while the user outputs σ, or $\perp$ to indicate error.*

– *bool* $\leftarrow \mathsf{AT.Vf}(\mathsf{vk}|\mathsf{sk}, t, [\mathsf{md}], \sigma)$, *the verification algorithm that takes as input either the public verification key* vk *or the private key* sk, *a token seed t, metadata* md *and the signature σ. It returns true if the token was valid.*

– $[\mathsf{ind} \leftarrow \mathsf{AT.ReadBit}(\mathsf{sk}, t, [\mathsf{md}], \sigma)]$, *the private bit extraction algorithm that takes as input the private key* sk *and token $(t, [\mathsf{md}], \sigma)$. It returns an indicator* $\mathsf{ind} \in \{\perp, 0, 1\}$ *which is either the private bit, or $\perp$.*

The notation of the above definition should be interpreted in a global sense. If one – for example – wants to use public metadata, it should be included everywhere it is mentioned. This listing then defines the following six notions:

1. With designated verification:
   (a) Anonymous single-use tokens
   (b) Anonymous single-use tokens with private metadata bit
   (c) Anonymous single-use tokens with public metadata
   (d) Anonymous single-use tokens with public and private metadata
2. With public verification:
   (a) Anonymous single-use tokens
   (b) Anonymous single-use tokens with public metadata

Examples of 1a and 1b are well known from previous work [DGS+18,KLOR20a]. A previous example of 2b is known as a partially blind signature scheme [AO00].

We will provide new examples of the last four (2a is implicit in 2b) in Section 4. We collectively refer to all of these as anonymous tokens.

We follow the convention of dividing the interactive protocol $\langle \mathsf{AT.User}, \mathsf{AT.Sign} \rangle$ into the non-interactive algorithms $\mathsf{AT.User}_0$, $\mathsf{AT.Sign}_0$ and $\mathsf{AT.User}_1$.

An anonymous token scheme must satisfy the following properties:

Definition 4 (Token correctness). *An anonymous token scheme* AT *is correct if any honestly generated token verifies. For any honestly generated* crs*,* $(\mathsf{pp}, \mathsf{sk}, [\mathsf{vk}])$*,* t *and* $[\mathsf{md}]$*,*

$$\Pr[\mathsf{AT.Vf}(\mathsf{vk}, t, [\mathsf{md}], \langle \mathsf{AT.User}(\mathsf{pp}, [\mathsf{vk}], t, \mathsf{md}), \mathsf{AT.Sign}(\mathsf{sk}, [\mathsf{md}], [b]) \rangle) = 1] = 1 - \mathsf{negl}(\lambda).$$

We split correctness of the private metadata bit into a separate definition in order to reduce notational clutter. This definition only applies in the private-key setting, and the parameters have been fixed accordingly.

Definition 5 (Correct private bit). *An anonymous token scheme* AT *is correct with respect to private metadata if the correct bit is retrieved successfully:*

$$\Pr[\mathsf{AT.ReadBit}(\mathsf{sk}, t, \langle \mathsf{AT.User}(\mathsf{pp}, t, [\mathsf{md}]), \mathsf{AT.Sign}(\mathsf{sk}, [\mathsf{md}], b) \rangle) = b] = 1 - \mathsf{negl}(\lambda).$$

No adversary should be able to redeem other tokens than those that have been correctly issued. The *one-more unforgeability* notion has become the common notion for anonymous credentials. It allows the adversary to claim ℓ tokens from the issuer, and the adversary should not be able to redeem $\ell + 1$ tokens. We require the tokens to be unique with respect to the value of the seed t.

```
Game $\mathrm{OMUF}_{\mathsf{AT},\mathcal{A},\ell}(\lambda)$
    $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{AT.Setup}(1^\lambda)$
    $(\mathsf{pp}, \mathsf{sk}, [\mathsf{vk}]) \leftarrow \mathsf{AT.KGen}(\mathsf{crs})$
    for $(b \in \{0,1\}, \mathsf{md}), q_{b,\mathsf{md}} := 0$
    $(t_i, \mathsf{md}_i, \sigma_i)_{i \in [\ell+1]} \leftarrow \mathcal{A}^{\textsc{Sign},\textsc{Verify},\textsc{Read}}(\mathsf{crs}, \mathsf{pp})$
    return $(\forall b \in \{0,1\}\ \forall \mathsf{md}, q_{b,\mathsf{md}} \leq \ell$ and
        $\forall i \neq j$ in $[\ell+1]\ (t_i, \mathsf{md}_i, \sigma_i) \neq (t_j, \mathsf{md}_j, \sigma_j)$
     and $\exists (b, \mathsf{md}) \in \{0,1\} \times \{\mathsf{md}\} : \forall i \in [\ell+1]$,
            $\mathsf{AT.ReadBit}(\mathsf{sk}, t_i, \sigma_i) = b$ and
            $\mathsf{AT.Vf}(\mathsf{sk}|\mathsf{vk}, t_i, [\mathsf{md}], \sigma_i) = \mathbf{true})$

Oracle $\textsc{Sign}(\mathsf{msg}, [\mathsf{md}], [b])$
    $q_{b,\mathsf{md}} := q_{b,\mathsf{md}} + 1$
    return $\mathsf{AT.Sign}_0(\mathsf{sk}, \mathsf{msg}, [\mathsf{md}], [b])$

Oracle $\textsc{Verify}(t, [\mathsf{md}], \sigma)$
    return $\mathsf{AT.Vf}(\mathsf{sk}|\mathsf{vk}, t, [\mathsf{md}], \sigma)$

Oracle $\textsc{Read}(t, \sigma)$
    return $\mathsf{AT.ReadBit}(\mathsf{sk}, t, [\mathsf{md}], \sigma)$
```

Fig. 3. One-more unforgeability with metadata.

```
Game $\mathrm{UNLINK}_{\mathsf{AT},\mathcal{A},m,[b],[\mathsf{md}]}(\lambda)$
$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{AT.Setup}(1^\lambda)$
$(\mathsf{st}, \mathsf{pp}, [\mathsf{vk}]) \leftarrow \mathcal{A}(\mathsf{crs}, [b], [\mathsf{md}])$
$q_0 := 0; q_1 := 0, \mathcal{Q} := \emptyset$
$(\mathsf{st}, (\mathsf{msg}_i)_{i\in\mathcal{Q}}) \leftarrow \mathcal{A}^{\mathrm{USER}_0,\mathrm{USER}_1}(\mathsf{st})$
if $\mathcal{Q} = \emptyset$ then return $0$
$j \leftarrow\$ \mathcal{Q}; \mathcal{Q} = \mathcal{Q} \setminus \{j\}$
$\sigma_j \leftarrow \mathsf{AT.User}_1(\mathsf{st}_j, [\mathsf{vk}], \mathsf{msg}_j, [\mathsf{md}])$
for $i \in \mathcal{Q}$
    $\sigma_i \leftarrow \mathsf{AT.User}_1(\mathsf{st}_i, [\mathsf{vk}], \mathsf{msg}_i, [\mathsf{md}])$
$\phi \leftarrow\$ \mathcal{S}_\mathcal{Q}$
$j' \leftarrow \mathcal{A}(\mathsf{st}, (t_j, \sigma_j), (t_{\phi(i)}, \sigma_{\phi(i)})_{i\in\mathcal{Q}})$
return $q_0 - q_1 \geq m$ and $j' = j$

Oracle $\mathrm{USER}_0()$
$q_0 := q_0 + 1$
$t_{q_0} \leftarrow\$ \{0,1\}^\lambda$
$(\mathsf{msg}_{q_0}, \mathsf{st}_{q_0}) \leftarrow \mathsf{AT.User}_0(\mathsf{pp}, [\mathsf{vk}], t_{q_0}, [\mathsf{md}'])$
$\mathcal{Q} := \mathcal{Q} \cup \{q_0\}$
return $(q_0, \mathsf{msg}_{q_0})$

Oracle $\mathrm{USER}_1(j, \mathsf{msg})$
if $j \notin \mathcal{Q}$ then
    return $\perp$
$\sigma \leftarrow \mathsf{AT.User}_1(\mathsf{st}_j, [\mathsf{vk}], \mathsf{msg}, [\mathsf{md}'])$
if $\sigma \neq \perp$ then
    $\mathcal{Q} := \mathcal{Q} \setminus \{j\}$
    $q_1 := q_1 + 1$
return $\sigma$
```

Fig. 4. Public-key unlinkability with fixed metadata. If X is a set, then $\mathcal{S}_X$ is the symmetric group of X.

Definition 6 (One-more unforgeability). *An anonymous token scheme* AT *is one-more unforgeable if for any* PPT *adversary $\mathcal{A}$, and any $\ell \geq 0$:*

$$\mathsf{Adv}^{\mathrm{omuf}}_{\mathsf{AT},\mathcal{A},\ell}(\lambda) := \Pr[\mathrm{OMUF}_{\mathsf{AT},\mathcal{A},\ell}(\lambda) = 1] = \mathsf{negl}(\lambda),$$

where $\mathrm{OMUF}_{\mathsf{AT},\mathcal{A},\ell}$ is the game defined in Figure 3.

Next, we want to provide user anonymity. The right notion for this is unlinkability, which guarantees that even colluding issuers and verifiers are unable to link tokens. Arbitrary metadata is a strong way of creating a link, and we omit this problem by only considering fixed public metadata for this notion. Notice that the adversary may query the user oracles for any public metadata md, but that we expect the post-processing to implicitly fail if $\mathsf{md} \neq \mathsf{md}'$. This is in line with for example expiry dates, which would otherwise have been solved in practice using key rotation, and the definition is (as usual) also using a fixed key. Private metadata is outside the control of the user, and gives one bit leakage. We fix it for this game. Note that the adversary controls the keys, and that we therefore do not need to provide access to signing and verification oracles.

Definition 7 (Unlinkability). *An anonymous token scheme* AT *is κ-unlinkable if for any* PPT *adversary $\mathcal{A}$, fixed b,* md*, and any $m > 0$,*

$$\mathsf{Adv}^{\mathrm{unlink}}_{\mathsf{AT},\mathcal{A},m,[b],[\mathsf{md}]}(\lambda) := \Pr\left[\mathrm{UNLINK}_{\mathsf{AT},\mathcal{A},m,[b],[\mathsf{md}]}(\lambda) = 1\right] \leq \frac{\kappa}{m} + \mathsf{negl}(\lambda),$$

where $\mathrm{UNLINK}_{\mathsf{AT},\mathcal{A},m}$ is the game defined in Figure 4.

```
Game $\mathrm{PMB}^{\beta}_{\mathsf{AT},\mathcal{A}}(\lambda)$
    $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{AT.Setup}(1^{\lambda})$
    $(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{AT.KGen}(\mathsf{crs})$
    $\beta' \leftarrow \mathcal{A}^{\textsc{Sign},\textsc{Sign}',\textsc{Verify}}(\mathsf{crs}, \mathsf{pp})$
    return $\beta'$

Oracle $\textsc{Sign}(\mathsf{msg}, [\mathsf{md}])$
    return $\mathsf{AT.Sign}_0(\mathsf{sk}, \mathsf{msg}, [\mathsf{md}], \beta)$

Oracle $\textsc{Sign}'(\mathsf{msg}, [\mathsf{md}], b)$
    return $\mathsf{AT.Sign}_0(\mathsf{sk}, \mathsf{msg}, [\mathsf{md}], b)$

Oracle $\textsc{Verify}(t, [\mathsf{md}], \sigma)$
    return $\mathsf{AT.Vf}(\mathsf{sk}, t, [\mathsf{md}], \sigma)$
```

Fig. 5. Game for private metadata bit for anonymous tokens.

We finally consider the private metadata bit. We give the adversary access to two signing oracles: One uses the adversary's chosen private bit, the other is using a fixed bit for the game. The adversary can also query a verification oracle. At the end, the adversary outputs its guess for the fixed challenge bit.

Definition 8 (Private metadata bit). *An anonymous token scheme* AT *provides private metadata bit if for any* PPT *adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathrm{pmb}}_{\mathsf{AT},\mathcal{A}}(\lambda) := \left|\Pr[\mathrm{PMB}^{0}_{\mathsf{AT},\mathcal{A}}(\lambda)] - \Pr[\mathrm{PMB}^{1}_{\mathsf{AT},\mathcal{A}}(\lambda)]\right| = \mathsf{negl}(\lambda),$$

where $\mathrm{PMB}^{\beta}_{\mathsf{AT},\mathcal{A}}$ *is the game defined in Figure 5.*

4 Anonymous Token Protocols

The Privacy Pass protocol [DGS+18] and its siblings [HIJ+21, KLOR20a] are based on Verifiable Oblivious Pseudo-Random Functions (VOPRF). Here, a user holds some secret input x and the signer holds a secret key k and they evaluate the function F obliviously such that the user learns $F(x, k)$ but nothing about k, and the signer learns nothing about the input x nor the output $F(x, k)$. Additionally, the user is ensured that the function is evaluated by the correct secret key.

We give three protocols for Anonymous Tokens (AT) with 1) public metadata, 2) public and private metadata, and 3) public metadata and public verifiability, respectively, constructed from the same framework.

At the core of our protocols lies a verifiable key transformation. Let $d := \mathtt{H}_m(\mathsf{md})$ and the curve point $U := [d]G + K$, where G is a public generator and K is the public key with a corresponding private key k. Let $e = (d + k)^{-1}$ be the new signing key and $W' = [e]T'$. Notice the relation

$$\mathsf{KT} : \log_G([d]G + K) = (d + k) = \log_{W'} T'. \qquad (1)$$

4.1 Secure Key Transformation

We argue that the key-transformation from k to e is secure against one-more unforgeability attacks. Several papers has been written using this transformation. Boneh and Boyen [BB04] shows that this transformation is secure against a non-adaptive attacker for arbitrary metadata md when used for signatures. Furthermore, Dodis and Yampolskiy [DY05] shows that this transformation is secure against active attackers when the set of possible metadata values is small, and give applications to PRFs. However, these works only prove security with respect to a fixed generators, while our construction signs arbitrary new generators in each execution of the protocol. Recently, Tyagi *et al.* [TCR+21] proved that this transformation is secure against an active attacker with respect to arbitrary generators and arbitrary set of metadata. They reduce the security of the transform to a new one-more gap strong inversion Diffie-Hellman problem (see Section 2.1). They also show that this new problem is equivalent to the simpler q-DL assumption. We summarize these results in a lemma.

Lemma 1. *Let* AT *be a scheme with keys* $(\mathsf{pk}, \mathsf{vk})$ *with security property* P *within adversarial advantage* $\mathsf{Adv}^{\mathrm{P}}_{\mathsf{AT},\mathcal{A}}(\lambda)$*, and assume we can prove the relation in Equation 1 within adversarial advantage* $\mathsf{Adv}^{\mathrm{rel}}_{\mathsf{KT},\mathcal{A}}(\lambda)$*. Then* $\mathcal{A}$ *has advantage* $\mathsf{Adv}^{\mathrm{P}}_{\mathsf{AT},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{rel}}_{\mathsf{KT},\mathcal{A}}(\lambda)$ *against property* P *in the scheme* AT *with transformed keys* $(\{e = (\mathsf{md} + \mathsf{sk})^{-1}, [e]G\})$.

4.2 Anonymous Tokens with Public Metadata

In Figure 6 we present an extension of Privacy Pass [DGS+18] with public metadata. The protocol is designated verifier, as the secret key is needed to verify tokens.

Setup and Key Generation. Let λ be the security parameter, let p be a prime and let E be an elliptic curve group of order p with generator G. Let $\mathtt{H}_t : \{0,1\}^* \to E$ and $\mathtt{H}_m : \{0,1\}^* \to \mathbb{Z}_p$ be hash functions, and assume that group elements and integers can be encoded uniquely as strings. Furthermore, let metadata md be an element of a public set of valid strings. Finally, let $\mathsf{sk} := k \leftarrow\!\$\ \mathbb{Z}_p^*$ be the signing key, and let $\mathsf{pk} := K := [k]G$ be the public key. We consider $G, E, p, \mathtt{H}_t, \mathtt{H}_m$ and K to be implicit knowledge in Figure 6.

Signing and Verification. The anonymous tokens protocol in Figure 6 uses the Π_{DLEQ}-protocol defined in Section 2.3. The signer computes a proof $\pi_{\mathtt{DLEQ}} := (c, z)$ of equality of discrete logarithms by instantiating the protocol $\Pi_{\mathrm{DLEQ}}(G, T', K, W'; e)$. Given the public parameters G and K, and $U := [d]G + K$, this is a proof that $\log_G U = d + k = \log_{W'} T'$. This proves that $W' = [e]T'$, where $e := (d + k)^{-1}$, is computed correctly with respect to d and K. To verify, the user instantiates the verification algorithm, denoted by $\mathtt{V}(\pi_{\mathtt{DLEQ}})$.

```
                                Signing
User(md, pk)                                      Signer(md, pk, sk)
$d := \mathtt{H}_m(\mathsf{md})$                  $d := \mathtt{H}_m(\mathsf{md})$
$U := [d]G + K$                                   $U := [d + k]G$
$t \leftarrow\$ \{0,1\}^\lambda, r \leftarrow\$ \mathbb{Z}_p^*$     $e := (d + k)^{-1}$
$T := \mathtt{H}_t(t)$
$T' := [r^{-1}]T$           $\xrightarrow{T'}$              $W' := [e]T'$
if not $\mathtt{V}(\pi_{\mathtt{DLEQ}})$   $\xleftarrow{W', \pi_{\mathtt{DLEQ}}}$   $\pi_{\mathtt{DLEQ}} \leftarrow \Pi_{\mathrm{DLEQ}}(G, T', K, W'; e)$
    return $\perp$
$W := [r]W'$
return $(t, \mathsf{md}, W)$
                                Redemption
User$(t, \mathsf{md}, W)$                          Verifier(sk)
                  $\xrightarrow{t, \mathsf{md}, W}$           $e := (\mathtt{H}_m(\mathsf{md}) + k)^{-1}$
                                                  if $W = [e]\mathtt{H}_t(t)$
                                                      return true
                                                  else
                                                      return false
```

Fig. 6. Designated verifier anonymous tokens with public metadata. Our protocol is a direct extension of Privacy Pass [DGS+18].

Theorem 1 (Completeness). *The anonymous token protocol with public metadata in Figure 6 is complete according to Definition 4.*

Proof. The completeness follows from expanding W:

$$W = [r]W' = [r][e]T' = [r][e][r^{-1}]T = [e]\mathtt{H}_t(t).$$

□

Theorem 2 (Unforgeability). *The anonymous token protocol with public metadata in Figure 6 achieve one-more unforgeability with respect to Definition 6.*

Proof. Using the key transformation as described in Lemma 1, the security of the protocol reduces to the security of the one-more gap strong inversion Diffie-Hellman game as shown in Figure 1. The advantage of an attacker follows from Definition 1 and is proven secure by Tyagi *et al.* [TCR+21, Theorem 1].

□

Theorem 3 (Unlinkability). *Fix metadata* md*. Within the set defined by all tokens using* md*, the anonymous token protocol with public metadata in Figure 6 achieve unlinkability with respect to Definition 7.*

Proof. This proof is identical to [DGS+18, Theorem 1]: As we sample $r \leftarrow\!\$\ \mathbb{Z}_p$ uniformly at random, it follows that our protocol is unconditionally unlinkable. Since T is a generator of E, then $T' = [r^{-1}]T$ is uniformly random and contain no information about t nor T. As the signer only sees T', and the verifier only receive t, and they are independent, there is no link between the view of the signer and the view of the verifier.

□

4.3 AT with Public and Private Metadata

In Figure 7, we present an extension of the PMBTokens [KLOR20a, Figure 8] with public metadata. This protocol is also designated verifier, requiring the secret key for verification.

Setup and Key Generation. Let λ be the security parameter, let p be a prime and let E be an elliptic curve group of order p with generators G_0, G_1. Let $\mathtt{H}_t : \{0,1\}^* \to E$, $\mathtt{H}_m : \{0,1\}^* \to \mathbb{Z}_p^*$ and $\mathtt{H}_s : \{0,1\}^* \to \mathbb{Z}_p^*$ be hash-functions, and assume that group elements and integers can be encoded uniquely as strings. Furthermore, let metadata md be an element of a public set of valid strings. Finally, let $\mathsf{sk} := (k_{0,0}, k_{0,1}, k_{1,0}, k_{1,1}) \leftarrow\!\$\ (\mathbb{Z}_p^*)^4$ (all $k_{i,j}$ being distinct) be the signing key, and let $\mathsf{pk} := \{K_{i,j}\} = \{[k_{i,j}]G_i\}$, for $i, j = 0, 1$, be the public key. This is implicit knowledge in the protocol description.

Signing and Verification. The anonymous tokens protocol in Figure 7 uses the $\mathit{\Pi}_{\mathrm{DLEQ2}}$-protocol defined in Section 2.4 twice as a subroutine to ensure that we afterwards can prove that the signed token W' was computed correctly. Given the generators G_0, G_1, T', S', the public keys $K_{i,j} := [k_{i,j}]G_i$ and the elements $V_i := [e_{i,0}]T' + [e_{i,1}]S'$, for $i, j = 0, 1$, we want to prove that the following relations hold:

$$\begin{bmatrix} G_0 + G_1 \\ V_i \end{bmatrix} = [e_{i,0}] \begin{bmatrix} [d]G_0 + K_{i,0} \\ T' \end{bmatrix} + [e_{i,1}] \begin{bmatrix} [d]G_1 + K_{i,1} \\ S' \end{bmatrix}.$$

We instantiate $\mathit{\Pi}_{\mathrm{DLEQ2}}(G_0, G_1, V_0; e_{0,0}, e_{0,1})$ and $\mathit{\Pi}_{\mathrm{DLEQ2}}(G_0, G_1, V_1; e_{1,0}, e_{1,1})$ in Figure 7 to get proofs $\pi_{\mathtt{AND}}$ and $\pi'_{\mathtt{AND}}$. We denote the verification by $\mathtt{V}(\pi_{\mathtt{AND}}, \pi'_{\mathtt{AND}})$.

We also use the $\mathit{\Pi}_{\mathrm{DLEQOR2}}$-protocol defined in Section 2.5. The signer computes an $\mathtt{OR}$-proof of equality of discrete logs by instantiating the zero-knowledge protocol $\mathit{\Pi}_{\mathrm{DLEQOR2}}(G_0, G_1, T', S', V_0, V_1, W'; e_{0,0}, e_{0,1}, e_{1,0}, e_{1,1})$. Consider the generators G_0, G_1, T', S', hashed metadata d and computed value W'. The signer then proves that W' is correctly computed, with respect to T' and S', and in

```
                                   Signing

User(md)                                              Signer(md, b, sk)
$d := \mathtt{H}_m(\mathsf{md})$                                     $d := \mathtt{H}_m(\mathsf{md})$
$t \leftarrow\$ \{0,1\}^\lambda, r \leftarrow\$ \mathbb{Z}_p^*$                      $e_{0,0} := (d + k_{0,0})^{-1}, e_{0,1} := (d + k_{0,1})^{-1}$
$T := \mathtt{H}_t(t)$                                     $e_{1,0} := (d + k_{1,0})^{-1}, e_{1,1} := (d + k_{1,1})^{-1}$
$T' := [r^{-1}]T$            $\xrightarrow{T'}$              $s \leftarrow\$ \{0,1\}^\lambda, S' := \mathtt{H}_s(T' || \mathsf{md} || s)$
                                                      $V_0 := [e_{0,0}]G_0 + [e_{0,1}]G_1$
                                                      $V_1 := [e_{1,0}]G_0 + [e_{1,1}]G_1$
                                                      $W' := [e_{b,0}]T' + [e_{b,1}]S'$
                                                      $\pi_{\mathtt{AND}} \leftarrow \Pi_{\mathrm{DLEQ2}}(G_0, G_1, V_0; e_{0,0}, e_{0,1})$
                                                      $\pi'_{\mathtt{AND}} \leftarrow \Pi_{\mathrm{DLEQ2}}(G_0, G_1, V_1; e_{1,0}, e_{1,1})$
$S' := \mathtt{H}_s(T' || \mathsf{md} || s)$   $\xleftarrow[\pi_{\mathtt{AND}}, \pi'_{\mathtt{AND}}, \pi_{\mathtt{OR}}]{s, V_0, V_1, W',}$   $\pi_{\mathtt{OR}} \leftarrow \Pi_{\mathrm{DLEQOR2}}(G_0, G_1, T', S', V_0, V_1, W'; e_{b,0}, e_{b,1})$
if not $\mathtt{V}(\pi_{\mathtt{AND}}, \pi'_{\mathtt{AND}}, \pi_{\mathtt{OR}})$
    return $\perp$
$S := [r]S'$
$W := [r]W'$
return $(t, \mathsf{md}, S, W)$

                                   Redemption

User(t, md, S, W)                                     Verifier(sk)
                             $\xrightarrow{t, \mathsf{md}, S, W}$          $T := \mathtt{H}_t(t), d := \mathtt{H}_m(\mathsf{md})$
                                                      $e_{0,0} := (d + k_{0,0})^{-1}, e_{0,1} := (d + k_{0,1})^{-1}$
                                                      $e_{1,0} := (d + k_{1,0})^{-1}, e_{1,1} := (d + k_{1,1})^{-1}$
                                                      $W_0 := [e_{0,0}]T + [e_{0,1}]S$
                                                      $W_1 := [e_{1,0}]T + [e_{1,1}]S$
                                                      if $W = W_0$ and $W \neq W_1$
                                                          return 0
                                                      if $W \neq W_0$ and $W = W_1$
                                                          return 1
                                                      else return $\perp$
```

Fig. 7. Designated verifier anonymous tokens with public and private metadata, an adjusted extension of Kreuter *et al.* [KLOR20a].

the same way as one of the committed values V_0 or V_1, with respect to G and H. That is, for either $b = 0$ or $b = 1$:

$$V_b = [e_{b,0}]G_0 + [e_{b,1}]G_1 \quad \wedge \quad W' = [e_{b,0}]T' + [e_{b,1}]S'.$$

We denote the verification of the proof $\pi_{\mathtt{OR}}$ by $\mathtt{V}(\pi_{\mathtt{OR}})$.

Theorem 4 (Completeness). *The anonymous token protocol with public and private metadata in Figure 7 is complete according to Definition 4 and will, according to Definition 5, return the correct metadata bit except with negligible probability.*

Proof. If the user submits (t, md, S, W), completeness follows from expanding W_b:

$$\begin{aligned} W_b &= [r]([e_{b,0}]T' + [e_{b,1}]S') = [r]([e_{b,0}][r^{-1}]T + [e_{b,1}]S') \\ &= [e_{b,0}]T + [r][e_{b,1}]S' = [e_{b,0}]\mathtt{H}_t(t||\mathsf{md}) + [e_{b,1}]S. \end{aligned}$$

Furthermore, the probability that this equation holds for both $b = 0$ and $b = 1$ is negligible. If that was the case, then

$$[e_{0,0}]T + [e_{0,1}]S = [e_{1,0}]T + [e_{1,1}]S.$$

As we require all keys $k_{i,j}$ to be distinct, it follows that all $e_{i,j}$ are distinct. Then, we have that

$$T = \left[\frac{e_{1,1} - e_{0,1}}{e_{0,0} - e_{1,0}}\right] S.$$

Since $T = \mathtt{H}_t(t)$ is sampled independently and uniformly at random, the probability that this equation holds is $1/p$, which is negligible.

□

Theorem 5 (Unforgeability). *The anonymous token protocol with public and private metadata in Figure 7 achieves one-more unforgeability with respect to Definition 6.*

Proof. For fixed metadata md we let the adversary query the signing oracle ℓ times for both $b = 0$ and $b = 1$. Using the key transformation as described in Lemma 1, the security of the protocol reduces to the security of the one-more gap strong inversion Diffie-Hellman game as shown in Figure 1. The advantage of an attacker follows from Definition 1 and is proven secure by Tyagi *et al.* [TCR+21, Theorem 1].

□

Theorem 6 (Unlinkability). *Fix private metadata b and public metadata* md. *Within the set defined by all tokens using b and* md, *the anonymous token protocol with public and private metadata in Figure 7 achieves unlinkability with respect to Definition 7.*

Proof. We note that it is easy to create many different anonymity sets to distinguish users based on private metadata being $b = 0$ or $b = 1$, and in combination with different values of public metadata md. We restrict the unlinkability to hold for users within the same anonymity sets based on b and md, both sampled according to the real distribution of private and public metadata. Let $\mathfrak{U}_{b,\mathsf{md}}$ be this set, and select two sessions from $\mathfrak{U}_{b,\mathsf{md}}$. Then it follows directly from [KLOR20a, Theorem 9] that the probability of success of the adversary will be upper bounded by $2/m + \mathsf{negl}(\lambda)$.

$\square$

Theorem 7 (Private metadata bit). *The anonymous token protocol with public and private metadata in Figure 7 provides private metadata bit with respect to Definition 8.*

Proof. This statement follows directly from the proof of [KLOR20a, Theorem 10], which describes a hybrid argument to prove that instances with private bit 0 are indistinguishable from instances with private bit 1. Notice in particular that the extra OR-proofs in our protocol are independent of the private bit b, and therefore need no additional simulation.

$\square$

4.4 Public Verifiability from Pairings

The authors of Privacy Pass [DGS$^+$18] described an application where the issuer and the recipient of a token would be the same entity, possibly separated by time. For the application we present in Section 6, those two roles are in fact separate, and one should therefore have a scheme that supports public verifiability. It remains an open problem to achieve this without pairings, unless we allow for two rounds of communication [AO00, WSMZ06].

We move on to provide a new variant of a partially blinded signature by Zhang, Safavi-Naini and Susilo [ZSS03]. The protocol allows a user and a signer to generate a signature on a user-private message m and agreed-upon metadata md. Both the issuance protocol and the signature consists of a single curve point.

We show that the idea underlying this scheme can be viewed as a combination of Boneh-Lynn-Shacham signatures [BLS01] and Privacy Pass, inheriting its attractive properties from both.

Setup and Key Generation. Let λ be the security parameter, let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing, where G_1, G_2 and g_T are generators for their respective prime p order groups. Furthermore, let $\mathtt{H}_1 : \{0,1\}^* \to \mathbb{G}_1$ and $\mathtt{H}_m : \{0,1\}^* \to \mathbb{Z}_p^*$ be hash functions, and assume that group elements and integers can be encoded uniquely as strings. Also, let md be an element of a public set of valid metadata strings. Finally, let $\mathsf{sk} := k \leftarrow\!\$\ \mathbb{Z}_p^*$ be the signing key, and let $\mathsf{pk} := K = [k]G_2$ be the public key. This is implicit knowledge in the protocol description.

Signing and Verification. Recall that the BLS-scheme signs a message m by hashing it to the group generated by G_1 and multiplying it with the secret key k; $W := [k]\mathtt{H}_1(m)$. The signature can then be verified by checking that

$$\hat{e}(\mathtt{H}_1(m), K) = \hat{e}(W, G_2).$$

Correctness follows from the linearity of the pairing.

We replace m by a token seed t, and use the same trick as earlier to concurrently update the key-pair based on metadata. Then we get the following anonymous token scheme:

Signing The user sends $T' := [r^{-1}]\mathtt{H}_1(t)$ to the issuer, who returns $W' := [e]T'$, for $e = (d+k)^{-1}$. The user can verify that the signature is correct by checking $\hat{e}(W', U) = \hat{e}(T', G_2)$, for $U := [d+k]G$, and then storing $(t, \mathsf{md}, W = [r]W')$.
Verification The user sends (t, md, W), and the recipient can verify the token by checking if $\hat{e}(W, U) = \hat{e}(T, G_2)$.

This scheme hides the token similarly to Privacy Pass, it can be verified without using the private key, and its unforgeability follows directly from BLS. We note that the check $\hat{e}(W', U) = \hat{e}(T', G_2)$ ensures that the tokens are signed correctly with respect to the public key. The complete protocol is listed in Figure 8. Finally, we note that we can batch-verify n tokens under the same key and metadata and check for equality by computing

$$\hat{e}\left(\sum_i [c_i]W_i, U\right) = \hat{e}\left(\sum_i [c_i]T_i, G_2\right),$$

where $c_1, \ldots, c_n$ are random coefficients. This saves the verifier of $2(n-1)$ expensive pairing-computations, which is especially useful in systems with large anonymity sets. Note that the verifier computes T_i from the received pre-tokens t_i, making sure that (T_i, W_i) is not just a scaling of a different valid token.

Theorem 8 (Completeness). *The anonymous token protocol with public metadata and public verifiability in Figure 8 is complete according to Definition 4.*

Proof. Completeness follows from expanding $\hat{e}(W, U)$:

$$\begin{aligned}\hat{e}(W, U) &= \hat{e}([r]W', [d+k]G_2) = \hat{e}([r][e]T', [d+k]G_2) \\ &= \hat{e}([r][e][r^{-1}]T, [d+k]G_2) = \hat{e}([e]T, [d+k]G_2) \\ &= \hat{e}(T, G_2)^{e\cdot(d+k)} = \hat{e}(T, G_2).\end{aligned}$$

□

Theorem 9 (Unforgeability). *The anonymous token protocol with public metadata and public verifiability in Figure 8 achieve one-more unforgeability with respect to Definition 6.*

```
                                  Signing

Client(md)                                               Signer(md, sk)
$d := \mathtt{H}_m(\mathsf{md})$                         $d := \mathtt{H}_m(\mathsf{md})$
$t \leftarrow\!\!\$ \{0,1\}^\lambda, r \leftarrow\!\!\$ \mathbb{Z}_p^*$    $e := (d + k)^{-1}$
$T := \mathtt{H}_1(t)$
$T' := [r^{-1}]T$
$U := [d]G_2 + K$           $\xrightarrow{T'}$           $W' := [e]T'$
                            $\xleftarrow{W'}$
if not $\hat{e}(W', U) = \hat{e}(T', G_2)$
    return $\perp$
$W := [r]W'$
return $(t, \mathsf{md}, W)$

                                  Redemption

User$(t, \mathsf{md}, W)$                                Verifier$(k)$
                            $\xrightarrow{t, \mathsf{md}, W}$    $T := \mathtt{H}_1(t)$
                                                         $U := [\mathtt{H}_m(\mathsf{md})]G_2 + K$
                                                         if $\hat{e}(W, U) = \hat{e}(T, G_2)$
                                                             return true
                                                         else
                                                             return false
```

Fig. 8. Anonymous tokens with public metadata and public verifiability by adjusting Zhang *et al.* [ZSS03] for asymmetric pairings.

Proof. Assume that we have an adversary who breaks unforgeability. In particular, this means that they can produce $\ell+1$ distinct and valid tuples $(t_i, \mathsf{md}, \sigma_i)$ but only query the signing oracle at most ℓ times. We use this adversary to construct an adversary against the (m,n)-OM-GAP-SDHI problem in the Random Oracle Model.

$\mathcal{A}_1^{\text{om-gap-sdhi}}$ Recall that we assume that the user and the signer agrees on the metadata. Run $\mathcal{O}_m$ on all acceptable metadata values to get the list $\{c_i\}$, and return it.

$\mathcal{A}_2^{\text{om-gap-sdhi}}$ Receive the input $G, K = [k]G, [y_i]G$. Set $\mathsf{vk} = K$ and the other parameters appropriately. Reprogram $\mathcal{O}_1$ such that it on input t returns $[t][y_j]G$ for the next j, which looks like a random group element. Whenever the adversary queries the SIGN oracle, forward the query to the SDH oracle. If the adversary wins the OMUF game for some metadata value md corresponding to an index γ, we have $\ell+1$ signatures $\sigma_i = [e_{\mathsf{md}}]\mathcal{O}_1(t_i) = [e_{\mathsf{md}}][t_i][y_1]G$. Use the programming of $\mathcal{O}_1$ to return $(\gamma, ([t^{-1}]\sigma_i, \alpha_i))$.

The result from $\mathcal{A}_2^{\text{om-gap-sdhi}}$ satisfies the (m,n)-OM-GAP-SDHI conditions.

One can construct a more detailed proof along the lines of [TCR+21, Appendix B] in order to get concrete bounds.

□

Theorem 10 (Unlinkability). *Fix metadata* md*. Within the set defined by all tokens using* md*, the anonymous token protocol with public metadata and public verifiability in Figure 8 achieve unlinkability with respect to Definition 7.*

Proof. Observe that given any valid token (t, md, W) and any honestly generated view (T', W') there exists a unique value r' such that both $W - [r']W'$ and $T - [r']T'$ holds, and hence, T is independent of any W. It follows that the anonymous token is unlinkable.

□

5 Performance and Comparison

In this section, we briefly describe the most efficient anonymous single-use token protocols with public metadata in the literature, for example, to enable batched revocation. We only consider protocols with one round of communication. We compare the protocols with our schemes in Table 1. To streamline the comparison, we assume that all parties know the public metadata, for example that md is the current date, and assume that this implicit knowledge is not sent. We instantiate the schemes with $\lambda = 128$ bits of security. Finally, we present a concrete example to show that we can replace DIT with our protocol in Figure 6 to improve both communication size and computational efficiency.

5.1 Anonymous single-use Tokens with Public Metadata

Privacy Pass. Our protocol in Figure 6 is inspired by Privacy Pass [DGS$^+$18], and they have identical structure and communication. The main difference is the change of private key used for signing, and the updated zero-knowledge proof with respect to the new public key, both depending on the public metadata. The zero-knowledge proofs are of the same size, and it follows that the communication sizes are equal. However, Privacy Pass does not allow public metadata unless we have one public key for each valid string of metadata, and hence, to allow for 2^N possible messages md, Privacy Pass must publish 2^N public keys.

DIT: De-Identified Authenticated Telemetry at Scale. DIT [HIJ$^+$21] is also inspired by Privacy Pass [DGS$^+$18], but uses an attribute-based VOPRF to generate new public keys on the fly. To allow for 2^N strings of public metadata, there are two main differences: 1) the public key consists of $N+2$ group elements, and 2) the token consists of an additional N group elements and zero-knowledge proofs to ensure that the correct public key is used in the signature.

Tokens from RSA. Abe and Fujisaki [AF96] presents a partially blind signature scheme based on RSA. The public exponent e must be at least two bits longer than the public metadata, and we fix this to be of length 130 bits. The user updates the public key to $e_{\mathsf{md}} = e \cdot \tau(\mathsf{md})$, for a public formatting function τ, when they blind the message, and the signer updates the secret key $d_{\mathsf{md}} = (e \cdot \tau(\mathsf{md}))^{-1} \mod N$ when signing. Otherwise, the partially blind signature scheme [AF96] is similar to the blind signature by Chaum [Cha82].

Tokens with Private Metadata. Kreuter *et al.* [KLOR20a] presents an extension of Privacy Pass [DGS$^+$18] to include private metadata. They publish two public keys, and the signer proves in zero-knowledge that the token is signed with one of the corresponding private keys. To ensure metadata privacy, each token is randomized based on a fresh seed s that is given to the user, and hence, the signature consists of a seed, a group element, and a proof. The token consists of the initial seed t in addition to two group elements. Like Privacy Pass, this protocol must publish a new pair of public keys for each valid string of metadata.

5.2 Comparison

We present a comparison of schemes in Table 1, where we focus on communication complexity. We note that both RSA and pairing based cryptography is usually slower than elliptic curve cryptography, in addition to requiring larger parameters. We also note that the updated keys in our protocols are only dependent on the secret key and the metadata, and can often be pre-computed. We conclude that when allowing for batched token-revocation, our protocols are more efficient than the state of the art in all categories.

While RSA and elliptic curve cryptography are primitives implemented in all mainstream cryptographic libraries, there are few trustworthy implementations of pairings. Even though there exists a few implementations[7], they are mostly for academic use, maybe except for the implementation in Rust used by Zcash[8]. We refer to [TCR+21, Table 1] for a comparison in computation between some protocols.

Public Metadata (PM)	PubKey	Request	Signature	Token
Privacy Pass [DGS+18]	$257 \cdot 2^N$	257	769	385
DIT [HIJ+21]	$257 \cdot (N+2)$	257	$769 \cdot (N+1)$	385
Our scheme (Figure 6)	257	257	769	385
PM + Private Metadata	**PubKey**	**Request**	**Signature**	**Token**
Kreuter *et al.* [KLOR20a]	$514 \cdot 2^N$	257	1921	642
Our Scheme (Figure 7)	1028	257	3203	642
PM + Public Verifiability	**PubKey**	**Request**	**Signature**	**Token**
Abe and Fujisaki [AF96]	3202	3072	3072	3200
Our scheme (Figure 8)	763	382	382	510

Table 1. Size given in bits. We compare the schemes for 128 bits of security, allowing for 2^N strings md of metadata. Token seed t is of size 128 bits, and metadata md is implicit knowledge. Privacy Pass, DIT, Kreuter *et al.* and our protocols in Figure 6 and 7 are instantiated with curve x25519 [Ber05], Abe and Fujisaki is instantiated with RSA-3072 and our protocol in Figure 8 is instantiated with BLS12-381 [YCKS21].

5.3 Telemetry Collection in WhatsApp

DIT [HIJ+21] was designed to allow users of WhatsApp to anonymously report telemetry data to Facebook. We present a concrete comparison to our protocols in Table 2. Here, we assume that Facebook wants to update their public keys only once a year, rotate signing keys every day, and only sign one token per user each day. We fix a year and encode public metadata as strings "YYYY-MM-DD".

Privacy Pass [DGS+18] is very efficient in terms of communication, but requires one public key per day. Hence, the public key is of size 93805 bits over a year of 365 days, that is, approximately 12 KB. An alternative method to download all keys and store them until usage is to use a Merkle-tree for key-transparency and give paths corresponding to the current public key as a part of each signature. Then, the public key consists of the root of size 256 bits, while each signature consists of $\lceil \log_2(365) \rceil = 9$ hashes of 256 bits in addition to the public key, the token, and the zero-knowledge proof. We give both instantiations in the table, and denote the alternative protocol as Privacy Pass+.

Our scheme in Figure 6 has the smallest overall communication complexity of all schemes. It offers much smaller keys than Privacy Pass, and much smaller signatures than Privacy Pass+ and DIT, saving up to 90 % in communication. If all 2 billion users of WhatsApp report their telemetry every day, our scheme

[7] Pairings: hackmd.io/@zkteam/eccbench
[8] Zcash: github.com/zkcrypto/bls12_381

in Figure 6 would save more than 1.7 TB of communication for the Facebook servers on a daily basis compared to the current implementation of DIT.

Our scheme in Figure 8 offers similar improvements to communication, in addition to public verifiability using pairings, but at the cost of less standardized cryptography and less efficient computation.

Protocol	PubKey	Request	Signature	Token
Privacy Pass [DGS$^+$18]	93805	257	769	385
Privacy Pass+	256	257	3330	385
DIT [HIJ$^+$21]	2313	257	7690	385
Our scheme (Figure 6)	257	257	769	385
Our scheme (Figure 8)	763	382	382	510

Table 2. Size given in bits. We compare Privacy Pass, DIT, and the protocols in Figure 6 and Figure 8 with daily key-rotation in a year, signing one token at a time.

6 Application to Contact Tracing

As nations started adopting digital contact tracing during the COVID-19 pandemic, privacy experts warned that such systems could enable the collection of people's contact graphs. The dp^{3}t protocol [T$^+$20] was eventually adopted as the *de facto* method for digital contact tracing through its implementation and deployment in iOS and Android as the Exposure Notification System (ENS).

We provide a brief overview of the basic dp^{3}t idea in order to put our contribution into context. The protocol is instantiated on each participating phone, which generates a random key (Temporary Exposure Key, TEK) every day. The TEK is used to generate new Rotating Proximity Identifiers (RPI) every 10–20 minutes, which is then broadcast from the phone using Bluetooth Low Energy (BLE). Other phones in the proximity store any RPI they hear.

If Alice tests positive for COVID-19 she can upload her TEKs (now renamed to diagnosis keys, DK) along with her BLE transmission strength to a health authority bulletin board. Bob's phone regularly checks the board to see if there is a sufficiently large overlap between published the DKs and the RPIs stored locally, and with sufficiently low difference between transmission strength and received strength. If this is the case, then Bob is given a suitable alert to let him know that he most likely has been in close vicinity of an infected individual, and should follow any advice given by the health authorities.

The process of uploading TEKs should depend on some sort of authorization. The dp^{3}t documentation describes a simplified model where a doctor receives the test results, and sends the patient an SMS with a short upload code. Now, this process may take precious person-hours during a pandemic. Some countries have therefore opted to connect their exposure notification with already existing centralized registries of positive test results, e.g., Norway, Denmark, and Estonia.

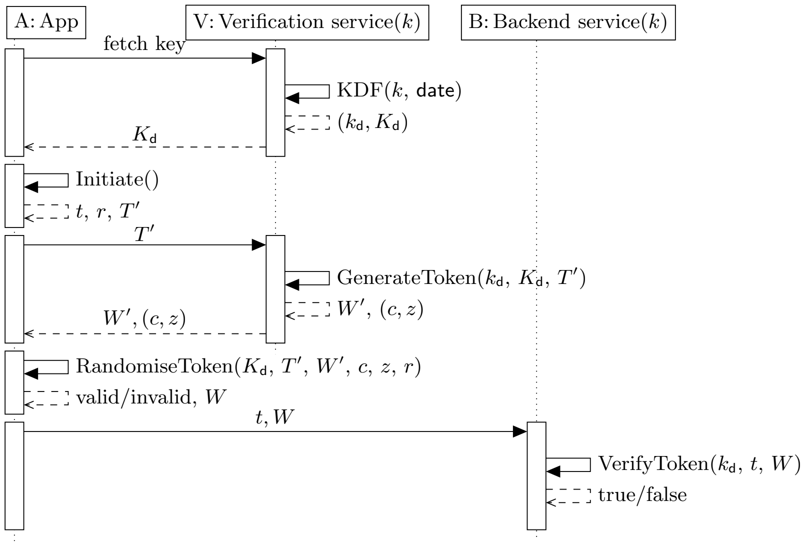

Fig. 9. A sequence diagram of anonymous tokens in the Norwegian app Smittestopp.

When starting the upload process, the user is prompted to log in to some government service ("verification"). Once the user has identified herself, the service makes a query to the relevant health registry. The service returns an access token to the app if there exists a recent positive test, which is then used to upload the keys to "backend". Unfortunately, this token may create an identifiable link from the meant-to-be-anonymous database of DKs, and unique identities in the health registry. Using anonymous single-use tokens, one can break this link (up to traffic analysis, e.g., logging timings and network addresses).

The Norwegian Institute of Public Health (NIPH) wanted the tokens to be timestamped in order to avoid users posting severely delayed keys: this would have allowed an attacker to get well again, move back out among other people, and only then upload to the backend service. Notice that merely tying the token to keys – e.g., by using a hash of the TEKs as the token seed t – would not avoid this attack, as those could have been generated and stored until the time of the attack. As a result, it was decided that the keys should be rotated regularly.

The original Privacy Pass protocol was reimplemented as a reusable C# package, to ease the integration into the Norwegian contact tracing app Smittestopp. The verification and backend services keep a master secret key k, and generate daily keys from some $\mathsf{KDF}(k, \mathsf{date})$. The public key is posted from the verification service. The full integration of anonymous tokens is described in Figure 9.

We finally note that this key distribution method suffers from a potential attack by a dishonest verification service that could serve special public keys to track individuals. It is, however, detectable by the users if they share their view of the public keys with each other to ensure consistency. The current solution was accepted by all involved stakeholders due to limited time and a weighting of the practical risk against the potential reward. The challenges with respect to key-rotation and key-sharing strongly motivated the authors' work in Section 4.

7 Conclusion

In this work, we have updated the definitions for anonymous single-use tokens to also include public metadata, and we have constructed three protocols that satisfy these definitions. Additionally, we combine public metadata with either private metadata or public verifiability, and show that all instantiations are efficient in practice. For situations with frequent key-rotation, we show that our protocols can save up to 90 % in communication over the state of the art. Furthermore, our protocols fit nicely into the Privacy Pass framework, which makes it easy to incorporate our contributions in the ongoing standardization processes by IETF and W3C, solving an open problem.

We also provide a description of how anonymous one-time tokens can be used to improve the user's privacy in contact tracing applications, and implemented this into the solution used in Norway. The app has more than one million users at the time of writing[9]. As the Norwegian app is built on top of the same code base as the Danish app, we consider it to be easy to extend the adaption of anonymous tokens to their app, and most likely others as well.

We would also like to suggest new use-cases for anonymous tokens. For example, anonymous tokens can improve the privacy of users traveling with public transport. Bus or train companies may require patrons to verify their period tickets for each journey, perhaps primarily to analyze traffic data. However, this can easily reveal the routes of single users while traveling in-between their home and workplace, but also to the abortion clinic, their church or to a public demonstration etc. If all travelers with valid tickets are given a series of tokens (e.g., with public metadata being the date or week or month the ticket is valid), then these can be redeemed when boarding. This way, the companies get the statistics they are interested in, without invading the user's privacy. In general, any systems with leveled authenticated login but anonymous actions can make use of our protocols, e.g., systems with electronic locks that only care if the user has certain privileges or not. We also note that Tyagi *et al.* [TCR+21] detail applications of a construction similar to ours to reduce key management complexity in the OPAQUE password authenticated key exchange protocol, and to ensure stronger security for password breach alerting services.

Finally, we would like to see improvements in three directions. Firstly, the zero-knowledge proofs used by the anonymous tokens protocol with public and private metadata in Figure 7 are much larger than the ones by Kreuter *et al.* [KLOR20a], in contrast to our protocol with public metadata in Figure 6 achieving the exact same communication cost as Privacy Pass [DGS+18]. In

[9] Smittestopp: fhi.no/om/smittestopp/nokkeltall-fra-smittestopp, last accessed 2021-12-01.

particular, we would like to reduce the number of proofs and extra group elements in the protocol in Section 4.3 . Secondly, we would like to provide protocols free of zero-knowledge proofs, to reduce the communication and computational cost, as provided in [KLOR20a, Section 7]. Finally, we would like to extend our protocols to achieve post-quantum security, continuing the work by Albrecht *et al.* [ADDS19] on lattice-based protocols.

Acknowledgments. The authors are very grateful to Henrik Walker Moe (Bekk Consulting AS) for stellar collaboration during the C# implementation phase. The final integration into Smittestopp was primarily a collaboration between Henrik, Johannes Brodwall (Sopra Steria) and Sindre Møgster Braaten (NIPH), with the authors and others as close consultants. We thank Nirvan Tyagi, Sofia Celi, Thomas Ristenpart and Christopher Wood for pointing out a flaw in the security game and unforgeability proof in an earlier version of this paper. The second author is grateful to the students Teodor Dahl Knutsen and Tallak Mannum for many useful comments to an earlier version of the manuscript. We would also like thank the anonymous reviewers at PETS 2021 and Financial Crypto 2022 for their feedback which greatly improved the presentation of this paper.

References

ADDS19. Martin R. Albrecht, Alex Davidson, Amit Deo, and Nigel P. Smart. Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. Cryptology ePrint Archive, Report 2019/1271, 2019. `https://eprint.iacr.org/2019/1271`.

AF96. Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'96*, volume 1163 of *Lecture Notes in Computer Science*, pages 244–251. Springer, Heidelberg, November 1996.

AMO08. Norio Akagi, Yoshifumi Manabe, and Tatsuaki Okamoto. An efficient anonymous credential system. In Gene Tsudik, editor, *FC 2008: 12th International Conference on Financial Cryptography and Data Security*, volume 5143 of *Lecture Notes in Computer Science*, pages 272–286. Springer, Heidelberg, January 2008.

AO00. Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 271–286. Springer, Heidelberg, August 2000.

BB04. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer, Heidelberg, May 2004.

Ber05. Daniel J. Bernstein. Curve25519: high-speed elliptic curve cryptography, 2005. https://cr.yp.to/ecdh.html.

BL13. Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 1087–1098. ACM Press, November 2013.

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, Heidelberg, December 2001.

BLS03. Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer, Heidelberg, September 2003.

BMR$^+$17. Jonathan Burns, Daniel Moore, Katrina Ray, Ryan Speers, and Brian Vohaska. EC-OPRF: Oblivious pseudorandom functions using elliptic curves. Cryptology ePrint Archive, Report 2017/111, 2017. `https://eprint.iacr.org/2017/111`.

BNPS02. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme. In Paul F. Syverson, editor, *FC 2001: 5th International Conference on Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 319–338. Springer, Heidelberg, February 2002.

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, Heidelberg, January 2003.

BPV12. Olivier Blazy, David Pointcheval, and Damien Vergnaud. Compact round-optimal partially-blind signatures. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12: 8th International Conference on Security in Communication Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 95–112. Springer, Heidelberg, September 2012.

CG08. Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008: 15th Conference on Computer and Communications Security*, pages 345–356. ACM Press, October 2008.

Cha82. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology – CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

Cha83. David Chaum. Blind signature system. In David Chaum, editor, *Advances in Cryptology – CRYPTO'83*, page 153. Plenum Press, New York, USA, 1983.

CHL05. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, Heidelberg, May 2005.

CHYC05. Sherman S. M. Chow, Lucas Chi Kwong Hui, Siu-Ming Yiu, and K. P. Chow. Two improved partially blind signature schemes from bilinear pairings. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP 05: 10th Australasian Conference on Information Security and Privacy*, volume 3574 of *Lecture Notes in Computer Science*, pages 316–328. Springer, Heidelberg, July 2005.

CKS09. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 481–500. Springer, Heidelberg, March 2009.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, Heidelberg, May 2001.

CL03. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, Heidelberg, September 2003.

CL04. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, Heidelberg, August 2004.

CMZ14. Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 1205–1216. ACM Press, November 2014.

CP93. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, Heidelberg, August 1993.

CPZ20. Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1445–1459. ACM Press, November 2020.

CV02. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, *ACM CCS 2002: 9th Conference on Computer and Communications Security*, pages 21–30. ACM Press, November 2002.

CZMS06. Xiaofeng Chen, Fangguo Zhang, Yi Mu, and Willy Susilo. Efficient provably secure restrictive partially blind signatures from bilinear pairings. In Giovanni Di Crescenzo and Avi Rubin, editors, *FC 2006: 10th International Conference on Financial Cryptography and Data Security*, volume 4107 of *Lecture Notes in Computer Science*, pages 251–265. Springer, Heidelberg, February / March 2006.

Dav21. Alex Davidson. Supporting the latest version of the privacy pass protocol. `https://blog.cloudflare.com/supporting-the-latest-version-of-the-privacy-pass-protocol`, 2021. (Accessed 01-December-2021).

DGS$^+$. Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: A privacy-enhancing protocol and browser extension. `https://privacypass.github.io`. (Accessed 01-December-2021).

DGS$^+$18. Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proceedings on Privacy Enhancing Technologies*, 2018(3):164–180, July 2018.

DY05. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, Heidelberg, January 2005.

FHKS16. Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model from weaker assumptions. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16: 10th International Conference on Security in Communication Networks*, volume 9841 of *Lecture Notes in Computer Science*, pages 391–408. Springer, Heidelberg, August / September 2016.

FHS15. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 233–253. Springer, Heidelberg, August 2015.

FIPR05. Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, Heidelberg, February 2005.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, Heidelberg, August 1987.

GMS10. Jorge Guajardo, Bart Mennink, and Berry Schoenmakers. Anonymous credential schemes with encrypted attributes. In Swee-Huay Heng, Rebecca N. Wright, and Bok-Min Goi, editors, *CANS 10: 9th International Conference on Cryptology and Network Security*, volume 6467 of *Lecture Notes in Computer Science*, pages 314–333. Springer, Heidelberg, December 2010.

Hen14. Ryan Henry. *Efficient Zero-Knowledge Proofs and Applications*. PhD thesis, University of Waterloo, 2014.

HG13. Ryan Henry and Ian Goldberg. Batch proofs of partial knowledge. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 502–517. Springer, Heidelberg, June 2013.

HIJ$^+$21. Sharon Huang, Subodh Iyengar, Sundar Jeyaraman, Shiv Kushwah, Chen-Kuei Lee, Zutian Luo, Payman Mohassel, Ananth Raghunathan, Shaahid Shaikh, Yen-Chieh Sung, and Albert Zhang. Dit: De-identified authenticated telemetry at scale. Technical report, Facebook

Inc., https://research.fb.com/wp-content/uploads/2021/04/DIT-De-Identified-Authenticated-Telemetry-at-Scale_final.pdf, 2021.

HS21. Lucjan Hanzlik and Daniel Slamanig. With a little help from my friends: Constructing practical anonymous credentials. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21. Association for Computing Machinery, 2021.

Int21. Internet Engineering Task Force. Privacy pass datatracker. https://datatracker.ietf.org/wg/privacypass, 2021. (Accessed 01-Dec-2021).

IT21. Subodh Iyengar and Erik Taubeneck. Fraud resistant, privacy preserving reporting using blind signatures. https://github.com/siyengar/private-fraud-prevention, 2021. (Accessed 01-December-2021).

JKK14. Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 233–253. Springer, Heidelberg, December 2014.

JKX18. Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 456–486. Springer, Heidelberg, April / May 2018.

KLOR20a. Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous tokens with private metadata bit. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 308–336. Springer, Heidelberg, August 2020.

KLOR20b. Ben Kreuter, Tancrede Lepoint, Michele Orru, and Mariana Raykova. Efficient anonymous tokens with private metadata bit. Cryptology ePrint Archive, Report 2020/072, 2020. https://eprint.iacr.org/2020/072.

Oka93. Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer, Heidelberg, August 1993.

PS96. David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'96*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer, Heidelberg, November 1996.

PWH$^+$17. Dimitrios Papadopoulos, Duane Wessels, Shumon Huque, Moni Naor, Jan Včelák, Leonid Reyzin, and Sharon Goldberg. Making NSEC5 practical for DNSSEC. Cryptology ePrint Archive, Report 2017/099, 2017. https://eprint.iacr.org/2017/099.

PZ13. Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1.1 revision 3, 2013. https://www.microsoft.com/en-us/research/project/u-prove.

T$^+$20. Carmela Troncoso et al. Decentralized privacy-preserving proximity tracing. https://arxiv.org/abs/2005.12273, 2020.

TCR$^+$21. Nirvan Tyagi, Sofia Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A. Wood. A fast and simple partially oblivious prf, with applications. Cryptology ePrint Archive, Report 2021/864, 2021.

TPY^{+}19. Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019: 28th USENIX Security Symposium*, pages 1556–1571. USENIX Association, August 2019.

Wor21. World Wide Web Consortium. Trust Token API Explainer. `https://github.com/WICG/trust-token-api`, 2021. (Accessed 01-December-2021).

WSMZ06. Qianhong Wu, Willy Susilo, Yi Mu, and Fanguo Zhang. Efficient partially blind signatures with provable security. In Marina Gavrilova, Osvaldo Gervasi, Vipin Kumar, C. J. Kenneth Tan, David Taniar, Antonio Laganá, Youngsong Mun, and Hyunseung Choo, editors, *Computational Science and Its Applications - ICCSA 2006*, pages 345–354, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

YCKS21. S. Yonezawa, S. Chikara, T. Kobayashi, and T. Saito. Pairing-Friendly Curves. `https://tools.ietf.org/id/draft-yonezawa-pairing-friendly-curves-02.html`, 2021. (Accessed 01-December-2021).

ZSS03. Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. Efficient verifiably encrypted signature and partially blind signature from bilinear pairings. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology - INDOCRYPT 2003: 4th International Conference in Cryptology in India*, volume 2904 of *Lecture Notes in Computer Science*, pages 191–204. Springer, Heidelberg, December 2003.

Paper II

Lattice-Based Proof of Shuffle and Applications to Electronic Voting

Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde and Thor Tunge

Published at the Cryptographers' Track RSA Conference, CT-RSA 2021. The full version is available at: eprint.iacr.org/2021/338.pdf.

Lattice-Based Proof of Shuffle and Applications to Electronic Voting

Diego F. Aranha[1], Carsten Baum[1⋆], Kristian Gjøsteen[2], Tjerand Silde[2], and Thor Tunge[2]

[1] Aarhus University, Denmark
{dfaranha,cbaum}@cs.au.dk
[2] Norwegian University of Science and Technology, Norway
{kristian.gjosteen,tjerand.silde}@ntnu.no

Abstract. A verifiable shuffle of known values is a method for proving that a collection of commitments opens to a given collection of known messages, without revealing a correspondence between commitments and messages. We propose the first practical verifiable shuffle of known values for lattice-based commitments.

Shuffles of known values have many applications in cryptography, and in particular in electronic voting. We use our verifiable shuffle of known values to build a practical lattice-based cryptographic voting system that supports complex ballots. Our scheme is also the first construction from candidate post-quantum secure assumptions to defend against compromise of the voter's computer using return codes.

We implemented our protocol and present benchmarks of its computational runtime. The size of the verifiable shuffle is 22τ KB and takes time 33τ ms for τ voters. This is around 5 times faster and 40 % smaller per vote than the lattice-based voting scheme by del Pino et al. (ACM CCS 2017), which can only handle yes/no-elections.

Keywords: Lattice-Based Cryptography, Proof of Shuffle, Verifiable Encryption, Return Codes, Electronic Voting, Implementation

1 Introduction

A *verifiable shuffle of known values* is a method for proving that a collection of commitments opens to a given collection of known messages, without revealing exactly which commitment corresponds to which message.

One well-known approach is due to Neff [Nef01]: define two polynomials, one that has the known messages as its roots and another that has the values committed to as its roots. Since polynomials are stable under permutations of

⋆ This work was funded by the European Research Council (ERC) under the European Unions' Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO). Part of this work was done while visiting NTNU in Trondheim.

their roots, it is sufficient to prove that these two polynomials have the same evaluation at a randomly chosen point.

Proving that the second polynomial has a given evaluation at a given point could be done using multiplication and addition proofs on the commitments. Usually multiplication proofs for committed values are quite expensive, while it is somewhat cheap to do proofs of linear combinations of committed values with public coefficients. Following the idea of Neff, the determinant of a particular band matrix is the difference of the two polynomials, and we show that the polynomials are equal by showing that the columns of the matrix are linearly dependent.

1.1 Our Contribution

Verifiable shuffle of known values. Our main contribution is a verifiable shuffle of known values for lattice-based commitments. This is the first efficient construction from a candidate post-quantum secure assumption of such a primitive. As discussed above, our construction is based on techniques originating with Neff [Nef01], although there are a number of obstacles with this approach in the lattice-based setting, where we use the commitments of Baum *et al.* [BDL$^+$18].

First of all, many group-homomorphic commitment schemes allow either direct or very simple verification of arbitrary linear relations. No known commitment scheme secure under an assumption considered as post-quantum secure has a similar structure, which means that we must use adaptations of existing proofs for linear relations. Secondly, the underlying algebraic structure is a ring, not a field. Since we need certain elements to be invertible, we need to choose challenges from special sets of invertible elements, and carefully adapt the proof so that the correctness of the shuffle is guaranteed.

In order to make our construction practical, we use the Fiat-Shamir transform to make the underlying Zero-Knowledge proofs non-interactive. We want to stress that our proof of security only holds in the conventional Random Oracle Model, which is not a sound model when considering quantum adversaries. Constructing a post-quantum secure verifiable shuffle of known values is an interesting open problem.

Voting from lattices. Our second contribution is the first construction of a practical voting system that is suitable for more general ballots (such as various forms of ranked choice voting, perhaps in various non-trivial combinations with party lists and candidate slates) and that is secure under lattice-based assumptions.

We adopt an architecture very similar to deployed cryptographic voting systems [HR16, Gjø11]. The protocol works as follows:

- The voter's computer commits to the voter's ballot and encrypts an opening of the ballot. The commitment and ciphertext are sent to a ballot box.
- When counting starts, the ballot box removes any identifying material from the ciphertext and sends this to the shuffle server.

– The shuffle server decrypts the openings, verifies the commitments and outputs the ballots. It uses our verifiable shuffle of known values to prove that the ballots are consistent with the commitments.
– One or more auditors inspect the ballot box and the shuffle server.

For this to work, the voter's ciphertext must contain a valid opening of the voter's commitment. To achieve this, we use the verifiable encryption scheme of Lyubashevsky and Neven [LN17].

This architecture seems to be an acceptable trade-off between security and practicality. It achieves privacy for voters under the usual threat models, it provides cast-as-intended verification via return codes, it achieves coercion-resistance via revoting, and it achieves integrity as long as at least one auditor is honest. However, the architecture makes it difficult to simultaneously achieve privacy and universal verifiability. (We cannot simply publish the ballot box, the decrypted ballots and the shuffle proofs, because the shuffle server then learns who submitted which ballot, breaking privacy.) This is often not a significant problem, because coercion resistance requires keeping the decrypted ballots secret when so-called Italian attacks apply, and it is usually quite expensive to achieve universal verifiability without publishing the decrypted ballots. If Italian attacks do not apply or coercion resistance is otherwise not an issue, if one is willing to pay the price, it would be possible to distribute the decryption among two (or more) players by using nested encryption and nested commitments, after which everything could be published and universal verifiability is achieved. The cost is significant, though. Limited verifiability can be achieved in cheaper ways.

Voting with return codes. Our third contribution is the first construction of a voting system that supports so-called return codes for verifying that ballots have been cast as intended and that is based on a candidate post-quantum assumption.

One of the major challenges in using computers for voting is that computers can be compromised. Countermeasures such as Benaloh challenges do not work very well in practice, since they are hard to understand[3]. Return codes can provide integrity for voters with a fairly high rate of fraud detection [GL16]. Return codes do not work well with complex ballots, but our scheme could be modified to use return codes only for parts of a complex ballot.

We again use the commitments and verifiable encryption. The voter's computer commits to a pre-code and proves that this pre-code has been correctly computed from the ballot and some key material. It also verifiably encrypts an opening of this commitment. The pre-code is later decrypted and turned into a return code, which the voter can inspect.

Implementation of our voting scheme. Our fourth contribution is a concrete choice of parameters for the system along with a prototype implementation, demonstrating that the scheme is fully practical. We choose parameters in such

[3] Very few members of the International Association for Cryptologic Research use Benaloh challenges when casting ballots in their elections.

a way that arithmetic in the used algebraic structures can be efficiently implemented. This gives a fairly low computational cost for the scheme, so the limiting factor seems to be the size of the proofs. For elections with millions of voters, the total proof size will be measured in gigabytes, while systems based on discrete logarithms would produce much smaller proofs. Since we do not try to achieve universal verifiability, which means that proofs in our architecture are only handled by well-resourced infrastructure players, the proof size is unlikely to matter much. (If ordinary voters were to verify all the shuffle proofs, this would still not be infeasible, but it would be more of an issue.)

1.2 Related Work

Verifiable shuffles. The idea for a verifiable shuffle of known values that we use was introduced by Neff [Nef01]. Since [Nef01], there has been a huge body of work improving verifiable shuffles of ciphertexts, but not for constructions that use post-quantum assumptions.

Costa *et al.* [CMM19] use ZK proofs for lattice commitments to show a correct shuffle and re-randomization of a collection of ciphertexts. They also adopt some of the techniques from Neff, but instead of using a linear algebra argument they use multiplication proofs. This is conceptually simpler than our approach, but turns out to be less efficient even with the newer, improved multiplication proofs of [ALS20]. A related concept to the verifiable shuffle of known values is the decrypting mix-net [Cha81], which proves that the decryption of a collection of ciphertexts equals a given collection of messages. Decryption mix-nets can be very fast [BHM20], but these constructions provide guarantees of correct decryption only if at least one participant in the mix-net is honest at the time of decryption, unlike our approach which provides proper soundness even if both the ballot box and shuffle server are compromised at the time of decryption.

Candidate post-quantum cryptographic voting systems. There is a large body of academic work on cryptographic voting systems, and several systems have been deployed in practice in Europe in e.g. Estonia [HR16] and Norway [HR16,Gjø11], while Switzerland [LPT19] also planned to use an e-voting system. All of these systems make significant efforts to provide so-called cast-as-intended verification, to defend against compromise of the voter's computer. For lower-stakes elections, Helios [Adi08] has seen significant use. All of these systems have roughly the same architecture, and offer varying levels of verifiability. None of these systems are secure against quantum computers.

Many real-world political elections have ballots that are essentially very simple, such as a single yes/no question, or a t-out-of-n structure (even though many such races can be combined to form a visually and cognitively complex ballot). However, real-world voting systems can also have more complicated ballots that cannot be decomposed to a series of simple, independent races. For example, the Australian parliamentary ballot may encode a total order on all candidates in a district, and transferable votes make counting quite complex. While work has been done on homomorphic counting for such elections, the usual approach is to recover cleartext ballots and count them.

While it is a simple exercise to use existing theoretical constructions to build a candidate quantum-safe voting system similar to the above deployed systems, the problem is that these constructions are practically inefficient, either because they are too computationally expensive or the proofs used are too large to make verification of many such proofs practical.

del Pino *et al.* [dLNS17] gives a feasible construction that uses homomorphic counting, but it is only applicable to yes/no-elections (though it can be extended to 1 out of n elections, at some cost). The scheme also does not try to defend against compromise of the voter's computer, limiting its applicability. Chillotti *et al.* [CGGI16] proposed a system based on homomorphic counting, but using fully homomorphic encryption. Again, this only supports 1 out of n elections, and practical efficiency is unclear. Gjøsteen and Strand [GS17] proposed a method for counting a complex ballot using homomorphic encryption. However, their scheme is not complete and the size of the circuit makes the system barely practical.

As discussed above, existing verifiable shuffles for candidate post-quantum secure cryptosystems could be used for generic constructions. Costa et al. [CMM19] uses certain ZK proofs for lattice commitments to show a correct shuffle and re-randomization of a collection of ciphertexts. The bottleneck of their approach are the underlying rather inefficient ZK proofs. The faster construct by Strand [Str19] is too restrictive in the choice of plaintext domain. Even given that shuffle, these schemes still require a verifiable (distributed) decryption for lattice-based constructions. These, currently, do not exist.

1.3 Using Verifiable Shuffles of Known Values

There are many other applications of a verifiable shuffle of known values than the one that is presented in this work. We give a brief overview of three such applications.

Verifiable shuffles. One application is to build a verifiable shuffle of ciphertexts of a homomorphic encryption scheme. The idea is to provide a method for proving that one collection of ciphertexts is a re-randomization of a second collection of ciphertexts, without revealing the correspondence of the ciphertexts.

The ciphertexts of such schemes can be re-randomized by adding an encryption of 0. The idea is then to commit to each homomorphic encryption separately and use a proof of linearity to show that the committed value is the ciphertext, plus an encryption of 0. Depending on the commitment and homomorphic encryption scheme, this proof can be very efficient - in particular if proofs of correct encryption are "cheap" using the commitment scheme. Then, one can perform a proof of shuffle of known openings on these auxiliary commitments, which succeeds if a permutation of the re-randomized ciphertexts is revealed.

Verifiable shuffled decryption. A related concept is *verifiable shuffled decryption*, to prove that a collection of ciphertexts decrypt to a collection of messages.

For this, one would create auxiliary commitments to the decryptions of each ciphertext as well as make a commitment to the secret key. Using the homomorphism on the commitments, one can now show that each commitment indeed contains the correct plaintext. Then, one can apply the proof of verifiable shuffle of known values to reveal the openings of the commitments in shuffled form.

The efficiency of this approach depends again on the choice of commitment and encryption scheme, and how good they fit together in terms of homomorphically evaluating the decryption function.

Voting with less trust. The above two ideas can be used to construct a cryptographic voting system that does not rely on a single server which performs decryptions and shuffles together. Each voter would directly encrypt his ballot, instead of committing to it. These votes are then shuffled and re-randomized multiple times, before they are verifiably decrypted (using threshold decryption). Such a system would be less susceptible to attacks against the secrecy of the votes. The downside is that it needs efficient threshold verifiable decryption to protect the privacy of the ballot.

2 Preliminaries

2.1 Notation

If Φ is a probability distribution, then $z \xleftarrow{\$} \Phi$ denotes that z was sampled according to Φ. If S is a finite set, then $s \xleftarrow{\$} S$ denotes that s was sampled uniformly from the set S. The expressions $z \leftarrow xy$ and $z \leftarrow \texttt{Func}(x)$ denote that z is assigned the product of x and y and the value of the function `Func` evaluated on x, respectively.

For two matrices $\boldsymbol{A} \in S^{\alpha \times \beta}, \boldsymbol{B} \in S^{\gamma \times \delta}$ over an arbitrary ring S, we denote by $\boldsymbol{A} \otimes \boldsymbol{B} \in S^{(\alpha \cdot \gamma) \times (\beta \cdot \delta)}$ their tensor product, i.e. the matrix

$$\boldsymbol{B} = \begin{pmatrix} b_{1,1} & \dots & b_{1,\delta} \\ \vdots & \ddots & \vdots \\ b_{\gamma,1} & \dots & b_{\gamma,\delta} \end{pmatrix}, \qquad \boldsymbol{A} \otimes \boldsymbol{B} := \begin{pmatrix} b_{1,1} \cdot \boldsymbol{A} & \dots & b_{1,\delta} \cdot \boldsymbol{A} \\ \vdots & \ddots & \vdots \\ b_{\gamma,1} \cdot \boldsymbol{A} & \dots & b_{\gamma,\delta} \cdot \boldsymbol{A} \end{pmatrix}.$$

2.2 The Rings $\boldsymbol{R}$ and $\boldsymbol{R_p}$

Let $p, r \in \mathbb{N}^+$ and $N = 2^r$. Then we define the rings $R = \mathbb{Z}[X]/\langle X^N + 1 \rangle$ and $R_p = R/\langle p \rangle$, that is, R_p is the ring of polynomials modulo $X^N + 1$ with integer coefficients modulo a prime p. If p is congruent to $1 + 2\delta \bmod 4\delta$, for $N \geq \delta > 1$ powers of 2, then $X^N + 1$ splits into δ irreducible factors.

We define the norms of elements $f(X) = \sum \alpha_i X^i \in R$ to be the norms of the coefficient vector as a vector in $\mathbb{Z}^N$:

$$||f||_1 = \sum |\alpha_i| \qquad ||f||_2 = \left(\sum \alpha_i^2\right)^{1/2} \qquad ||f||_\infty = \max_{i \in \{1,\dots,N\}} \{|\alpha_i|\}.$$

For an element $\bar{f} \in R_p$ we choose coefficients as the representatives in $\left[-\frac{p-1}{2}, \frac{p-1}{2}\right]$, and then compute the norms as if $\bar{f}$ is an element in R. For vectors $\boldsymbol{a} = (a_1, \dots, a_k) \in R^k$ we define the 2-norm to be $\|\boldsymbol{a}\|_2 = \sqrt{\sum \|a_i\|^2}$, and analogously for the ∞-norm. We omit the subscript in the case of the 2-norm.

One can show that sufficiently short elements in the ring R_p (with respect to the aforementioned norms) are invertible.

Lemma 1 ([LS18], Corollary 1.2). *Let $N \geq \delta > 1$ be powers of 2 and p a prime congruent to $2\delta + 1$ mod 4δ. Then $X^N + 1$ factors into δ irreducible factors $X^{N/\delta} + r_j$, for some r_j's in R_p. Additionally, any non-zero y such that*

$$\|y\|_\infty < p^{1/\delta}/\sqrt{\delta} \quad \text{or} \quad \|y\| < p^{1/\delta}$$

is invertible in R_p.

For the remaining part of this paper we will assume that the parameters p, δ and N are chosen such that Lemma 1 is satisfied. We define a set of short elements

$$D_{\beta_\infty} = \{x \in R_p \mid \|x\|_\infty \leq \beta_\infty\}.$$

We furthermore define

$$\mathcal{C} = \{c \in R_p \mid \|c\|_\infty = 1, \|c\|_1 = \nu\},$$

which consists of all elements in R_p that have trinary coefficients and are non-zero in exactly ν positions, and we denote by

$$\bar{\mathcal{C}} = \{c - c' \mid c \neq c' \in \mathcal{C}\}$$

the set of differences of distinct elements in $\mathcal{C}$. The size of $\mathcal{C}$ is $2^\nu \binom{N}{\nu}$. It can be seen from Lemma 1 that, for a suitable choice of parameters, we can ensure that all non-zero elements from the three sets are invertible.

We need a bound on how many roots a polynomial can have over the ring R_p. The total number of elements in the ring is $|R_p| = p^N$.

Lemma 2. *Let $N \geq \delta \geq 1$ be powers of 2, p a prime congruent to $2\delta + 1$ mod 4δ and $T \subseteq R_p$. Let $g \in R_p[X]$ be a polynomial of degree τ. Then, g has at most τ^δ roots in T, and $\Pr[g(\rho) = 0 | \rho \xleftarrow{\$} T] \leq \tau^\delta / |T|$.*

Proof. First, by Lemma 1, we divide $X^N + 1$ into δ irreducible factors $X^{N/\delta} + r_j$. Each of the irreducible factors contributes at most τ roots to a polynomial $g \in R_p[X]$ of degree τ. Using the Chinese remainder theorem to combine the roots, we get that g has at most τ^δ roots in R_p. If we choose $\rho \xleftarrow{\$} R_p$ uniformly at random, the probability that this is a root of g is the total number of roots divided by the size of the ring. Since T is a subset of R_p, it can contain at most as many roots as R_p itself. □

2.3 The Discrete Gaussian Distribution

The continuous normal distribution over $\mathbb{R}^k$ centered at $\boldsymbol{v} \in \mathbb{R}^k$ with standard deviation σ is given by

$$\rho(\boldsymbol{x})^N_{\boldsymbol{v},\sigma} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-||\boldsymbol{x} - \boldsymbol{v}||^2}{2\sigma^2}\right).$$

When sampling randomness for our lattice-based commitment and encryption schemes, we'll need samples from the *discrete Gaussian distribution.* This distribution is achieved by normalizing the continuous distribution over R^k by letting

$$\mathcal{N}^k_{\boldsymbol{v},\sigma}(\boldsymbol{x}) = \frac{\rho^{kN}_{\boldsymbol{v},\sigma}(\boldsymbol{x})}{\rho^{kN}_{\sigma}(R^k)} \text{ where } \boldsymbol{x} \in R^k \text{ and } \rho^{kN}_{\sigma}(R^k) = \sum_{\boldsymbol{x} \in R^k} \rho^{kN}_{\sigma}(\boldsymbol{x}).$$

When $\sigma = 1$ or $\boldsymbol{v} = \boldsymbol{0}$, they are omitted.

2.4 Knapsack Problems

We first define the Search Knapsack problem in the ℓ_2 norm, also denoted as SKS^2. The SKS^2 problem is exactly the Module-SIS problem in its Hermite Normal Form.

Definition 1. *The* $\mathsf{SKS}^2_{n,k,\beta}$ *problem is to find a short vector* $\boldsymbol{y}$ *satisfying* $[\,\boldsymbol{I}_n \quad \boldsymbol{A}'\,] \cdot \boldsymbol{y} = \boldsymbol{0}^n$ *for a given random matrix* $\boldsymbol{A}'$*. An algorithm* $\mathcal{A}$ *has advantage* ϵ *in solving the* $\mathsf{SKS}^2_{n,k,\beta}$ *problem if*

$$\Pr\left[\begin{array}{c|c} \|y_i\|_2 \le \beta \wedge & \boldsymbol{A}' \leftarrow R_q^{n\times(k-n)}; \\ [\,\boldsymbol{I}_n \quad \boldsymbol{A}'\,] \cdot \boldsymbol{y} = \boldsymbol{0}^n & \boldsymbol{0} \neq \boldsymbol{y} = [y_1, \ldots, y_k]^\top \leftarrow \mathcal{A}(\boldsymbol{A}') \end{array}\right] \ge \epsilon.$$

Additionally, we define the Decisional Knapsack problem in the ℓ_∞ norm (DKS^∞). The DKS^∞ problem is equivalent to the Module-LWE problem when the number of samples is limited.

Definition 2. *The* $\mathsf{DKS}^\infty_{n,k,\beta}$ *problem is to distinguish the distribution* $[\,\boldsymbol{I}_n \quad \boldsymbol{A}'\,] \cdot \boldsymbol{y}$ *for a short* $\boldsymbol{y}$ *from the uniform distribution when given* $\boldsymbol{A}'$*. An algorithm* $\mathcal{A}$ *has advantage* ϵ *in solving the* $\mathsf{DKS}^\infty_{n,k,\beta}$ *problem if*

$$\Big|\Pr[b = 1 \mid \boldsymbol{A}' \leftarrow R_q^{n\times(k-n)}; \boldsymbol{y} \leftarrow D^k_\beta; b \leftarrow \mathcal{A}(\boldsymbol{A}', [\,\boldsymbol{I}_n \quad \boldsymbol{A}'\,] \cdot \boldsymbol{y})]$$
$$- \Pr[b = 1 \mid \boldsymbol{A}' \leftarrow R_q^{n\times(k-n)}; \boldsymbol{u} \leftarrow R^n_q; b \leftarrow \mathcal{A}(\boldsymbol{A}', \boldsymbol{u})]\Big| \ge \epsilon.$$

3 Lattice-Background: Commitments and ZK Proofs

We first introduce the commitments of Baum et al. [BDL+18], and continue with a zero-knowledge proof protocol of linear relation over the ring R_p using these commitments. The protocol is implicitly mentioned in [BDL+18].

3.1 Lattice-Based Commitments

Algorithms. The scheme consists of three algorithms: $\texttt{KeyGen}_{\text{C}}$, $\texttt{Com}$, and $\texttt{Open}$ for key generation, commitments and verifying an opening, respectively. We describe these algorithms for committing to one message, and refer to [BDL+18] for more details.

$\texttt{KeyGen}_{\text{C}}$ outputs a public matrix $\boldsymbol{B}$ over R_p of the form

$$\begin{aligned} \boldsymbol{B}_1 &= \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{B}'_1 \end{bmatrix} && \text{where}\, \boldsymbol{B}'_1 \xleftarrow{\$} R_p^{n\times(k-n)} \\ \boldsymbol{b}_2 &= \begin{bmatrix} \boldsymbol{0}^n & 1 & \boldsymbol{b}'_2 \end{bmatrix} && \text{where}\, (\boldsymbol{b}'_2)^\top \xleftarrow{\$} R_p^{(k-n-1)}, \end{aligned}$$

for width k and height $n+1$ of the public key $\texttt{pk} := \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{b}_2 \end{bmatrix}$.

$\texttt{Com}$ commits to messages $m \in R_p$ by sampling an $\boldsymbol{r}_m \xleftarrow{\$} D_{\beta_\infty}^k$ and computing

$$\texttt{Com}(m; \boldsymbol{r}_m) = \boldsymbol{B} \cdot \boldsymbol{r}_m + \begin{bmatrix} \boldsymbol{0} \\ m \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}_1 \\ c_2 \end{bmatrix} = [\![m]\!].$$

$\texttt{Com}$ outputs $[\![m]\!]$ and $d = (m; \boldsymbol{r}_m, 1)$.

$\texttt{Open}$ verifies whether an opening $(m; \boldsymbol{r}_m, f)$ with $f \in \bar{\mathcal{C}}$ is a valid opening of $\boldsymbol{c}_1, c_2$ by checking if

$$f \cdot \begin{bmatrix} \boldsymbol{c}_1 \\ c_2 \end{bmatrix} \stackrel{?}{=} \boldsymbol{B} \cdot \boldsymbol{r}_m + f \cdot \begin{bmatrix} 0 \\ m \end{bmatrix},$$

and that $\|\boldsymbol{r}_m[i]\| \leq 4\sigma_{\text{C}}\sqrt{N}$ for $i \in [k]$ with $\sigma_{\text{C}} = 11 \cdot \beta_\infty \cdot \nu \cdot \sqrt{kN}$. $\texttt{Open}$ outputs 1 if all these conditions hold, and 0 otherwise.

Baum et al. [BDL+18] proved the security properties of the commitment scheme with respect to the knapsack problems (which in turn are versions of standard Module-SIS/Module-LWE problems) defined in Section 2.4. More concretely, they showed that any algorithm $\mathcal{A}$ that efficiently solves the hiding property can be turned into an algorithm $\mathcal{A}'$ solving $\mathsf{DKS}^\infty_{n+1,k,\beta_\infty}$ with essentially the same runtime and success probability. Furthermore, any algorithm $\mathcal{A}$ that efficiently solves the binding problem can be turned into an algorithm $\mathcal{A}''$ solving $\mathsf{SKS}^2_{n,k,16\sigma_{\text{C}}\sqrt{\nu N}}$ with the same success probability.

The commitments [BDL+18] have a weak additively homomorphic property:

Proposition 1. *Let $\boldsymbol{z}_0 = \texttt{Com}(m; \boldsymbol{r}_m)$ be a commitment with opening $(m; \boldsymbol{r}_m, f)$ and let $\boldsymbol{z}_1 = \texttt{Com}(\rho; \boldsymbol{0})$. Then $\boldsymbol{z}_0 - \boldsymbol{z}_1$ is a commitment with opening $(m - \rho; \boldsymbol{r}_m, f)$.*

The proof follows from the linearity of the verification algorithm.

3.2 Zero-Knowledge Proof of Linear Relations

Let $[\![x]\!], [\![x']\!]$ be commitments as above such that $x' = \alpha x + \beta$ for some public $\alpha, \beta \in R_p$. Then Π_{Lin} in Figure 1 shows a zero-knowledge proof of knowledge (ZKPoK) of this fact (it is an adapted version of the linearity proof in [BDL+18]). The proof is a Σ protocol that aborts[4] with a certain probability to achieve the zero-knowledge property. For the protocol in Figure 1 we define

$$[\![x]\!] = \texttt{Com}(x; \boldsymbol{r}) = \begin{bmatrix} \boldsymbol{c}_1 \\ c_2 \end{bmatrix} \quad , \quad [\![x']\!] = \texttt{Com}(x'; \boldsymbol{r}') = \begin{bmatrix} \boldsymbol{c}_1' \\ c_2' \end{bmatrix} .$$

$$\begin{array}{lcl}
\underline{\text{Prover } \mathcal{P}} & & \underline{\text{Verifier } \mathcal{V}} \\
\boldsymbol{y}, \boldsymbol{y}' \xleftarrow{\$} \mathcal{N}^k_{\sigma_{\text{C}}} & & \\
\boldsymbol{t} \leftarrow \boldsymbol{B}_1 \boldsymbol{y}, \boldsymbol{t}' \leftarrow \boldsymbol{B}_1 \boldsymbol{y}' & & \\
u \leftarrow \alpha \langle \boldsymbol{b}_2, \boldsymbol{y} \rangle - \langle \boldsymbol{b}_2, \boldsymbol{y}' \rangle & \xrightarrow{\quad \boldsymbol{t}, \boldsymbol{t}', u \quad} & \\
 & \xleftarrow{\quad d \quad} & d \xleftarrow{\$} \mathcal{C} \\
\boldsymbol{z} \leftarrow \boldsymbol{y} + d\boldsymbol{r} & & \\
\boldsymbol{z}' \leftarrow \boldsymbol{y}' + d\boldsymbol{r}' & & \\
\text{Continue with probability:} & & \\
\displaystyle\prod_{(\boldsymbol{a},\boldsymbol{b}) \in \{(\boldsymbol{r},\boldsymbol{z}),(\boldsymbol{r}',\boldsymbol{z}')\}} \min\left(1, \frac{\mathcal{N}^k_{\sigma_{\text{C}}}(\boldsymbol{b})}{M \cdot \mathcal{N}^k_{d\boldsymbol{a},\sigma_{\text{C}}}(\boldsymbol{b})}\right) & \xrightarrow{\quad \boldsymbol{z}, \boldsymbol{z}' \quad} & \\
 & & \textbf{return } \text{Accept iff} \\
 & & 1: \quad \|\boldsymbol{z}[i]\|, \|\boldsymbol{z}'[i]\| \leq 2\sigma_{\text{C}}\sqrt{N}, \quad i \in [k] \\
 & & 2: \quad \boldsymbol{B}_1 \boldsymbol{z} \stackrel{?}{=} \boldsymbol{t} + d\boldsymbol{c}_1 \\
 & & 3: \quad \boldsymbol{B}_1 \boldsymbol{z}' \stackrel{?}{=} \boldsymbol{t}' + d\boldsymbol{c}_1' \\
 & & 4: \quad \alpha \langle \boldsymbol{b}_2, \boldsymbol{z} \rangle - \langle \boldsymbol{b}_2, \boldsymbol{z}' \rangle \stackrel{?}{=} (\alpha c_2 + \beta - c_2')d + u
\end{array}$$

Fig. 1: Protocol Π_{Lin} is a Sigma-protocol to prove the relation $x' = \alpha x + \beta$, given the commitments $[\![x]\!], [\![x']\!]$ and the scalars α, β.

In [BDL+18] the authors show that a version of Π_{Lin} is a Honest-Verifier Zero-Knowledge Proof of Knowledge for the aforementioned commitment scheme. This can directly be generalized to relations of the form $\alpha \cdot \tilde{x} + \beta$ as follows:

Lemma 3. *Let $\alpha, \beta, [\![x]\!], [\![x']\!]$ be defined as above. Then Π_{Lin} is a HVZK proof of the relation*

$$R_{\text{Lin}} = \left\{ (s, w) \middle| \begin{array}{l} s = (\alpha, \beta, [\![x]\!], [\![x']\!], \boldsymbol{B}_1, \boldsymbol{b}_2), w = (\tilde{x}, \tilde{\boldsymbol{r}}, \tilde{\boldsymbol{r}}', f), \\ \texttt{Open}([\![x]\!], \tilde{x}, \tilde{\boldsymbol{r}}, f) = \texttt{Open}([\![x']\!], \alpha \cdot \tilde{x} + \beta, \tilde{\boldsymbol{r}}', f) = 1 \end{array} \right\} .$$

[4] This approach is usually referred to as *Fiat Shamir with Aborts* (see e.g. [Lyu09, Lyu12] for a detailed description). If the proof is compiled with a random oracle into a NIZK, then these aborts only increase the prover time by a constant factor.

The proof for this is exactly the same as in [BDL$^+$18], and we do only sketch it now: assume that we can rewind an efficient poly-time prover and obtain two accepting transcripts with the same first message $\boldsymbol{t}, \boldsymbol{t}', u$ but differing $d, \overline{d}$ (as well as responses $\boldsymbol{z}, \boldsymbol{z}', \overline{\boldsymbol{z}}, \overline{\boldsymbol{z}'}$). Then one can extract valid openings $(\tilde{x}; \tilde{\boldsymbol{r}}, f)$ and $(\alpha\tilde{x} + \beta; \tilde{\boldsymbol{r}}', f)$ for $[\![x]\!], [\![x']\!]$ respectively as follows: From the two accepting transcripts and the equations checked by the verifier we can set $f = d - \overline{d}$, $\tilde{\boldsymbol{r}} = \boldsymbol{z} - \overline{\boldsymbol{z}}, \tilde{\boldsymbol{r}}' = \boldsymbol{z}' - \overline{\boldsymbol{z}'}$ where it must hold that

$$\alpha\langle \boldsymbol{b}_2, \tilde{\boldsymbol{r}}\rangle - \langle \boldsymbol{b}_2, \tilde{\boldsymbol{r}}'\rangle \overset{?}{=} f(\alpha c_2 + \beta - c_2').$$

By setting $\tilde{x} = c_2 - f^{-1}\langle \boldsymbol{b}_2, \tilde{\boldsymbol{r}}\rangle$ and $\tilde{x}' = c_2' - f^{-1}\langle \boldsymbol{b}_2, \tilde{\boldsymbol{r}}'\rangle$, we then have that $\alpha x + \beta = x'$ by the aforementioned equation. The validity and bounds of the opening follow from the same arguments as in [BDL$^+$18].

Compression. Using the techniques from [GLP12,BG14], as already mentioned in [BDL$^+$18, Section 5.3], allows to compress the non-interactive version of the aforementioned zero-knowledge proof. The main idea is that the prover only hashes the parts of the proof that got multiplied by the uniformly sampled part $\boldsymbol{B}_1'$ of $\boldsymbol{B}_1$, and that the verifier only checks an approximate equality with these when recomputing the challenge. We do the following changes to the protocol.

The prover samples vectors $\boldsymbol{y}, \boldsymbol{y}'$ of dimension $k - n$ according to σ_{C}, then computes $\boldsymbol{t} = \boldsymbol{B}_1'\boldsymbol{y}$ and $\boldsymbol{t}' = \boldsymbol{B}_1'\boldsymbol{y}'$. Note that u is computed as before, as the n first values of $\boldsymbol{b}_2$ are zero. Then $\boldsymbol{z}$ and $\boldsymbol{z}'$ are computed as earlier, but are of dimension $k - n$ instead of k. The prover computes the challenge d as

$$d = \mathtt{H}(\boldsymbol{B}, [\![x]\!], [\![x']\!], \alpha, \beta, u, \lfloor \boldsymbol{t} \rceil_\gamma, \lfloor \boldsymbol{t}' \rceil_\gamma),$$

where $\gamma \in \mathbb{N}$ and $\lfloor \cdot \rceil_\gamma$ denotes rounding off the least γ bits.

To make sure that the non-interactive proof can be verified, we must ensure that d can be re-computed from the public information. Let $\hat{\boldsymbol{t}} = \boldsymbol{B}_1'\boldsymbol{z} - d\boldsymbol{c}_1$ and $\hat{\boldsymbol{t}}' = \boldsymbol{B}_1'\boldsymbol{z}' - d\boldsymbol{c}_1'$ and observe that $\hat{\boldsymbol{t}}[i] - \boldsymbol{t}[i] = d\boldsymbol{r}[i]$, for each coordinate $i \in [n]$, and similar for $\hat{\boldsymbol{t}}'$ and $\boldsymbol{t}'$. For honestly generated randomness, for each $i \in [k]$, we have that $\|\boldsymbol{r}[i]\| \leq \beta_\infty\sqrt{N}$, and since $d \in \bar{\mathcal{C}}$, we have that $\|d\| = \sqrt{\nu}$. It follows that $\|d\boldsymbol{r}[i]\|_\infty \leq \beta_\infty\sqrt{\nu N}$, and similar for $d\boldsymbol{r}'[i]$. When hashing $\boldsymbol{t}$ and $\boldsymbol{t}'$ to get the challenge d, we then remove the $\gamma = \lceil \log \beta_\infty \sqrt{\nu N} \rceil$ lower bits of each coordinate first, to ensure that both the prover and the verifier compute on the same value. Hence, before outputting the proof, the prover will also test that

$$\begin{aligned} d' &= \mathtt{H}(\boldsymbol{B}, [\![x]\!], [\![x']\!], \alpha, \beta, \hat{u}, \lfloor \boldsymbol{B}_1'\boldsymbol{z} - d\boldsymbol{c}_1 \rceil_\gamma, \lfloor \boldsymbol{B}_1'\boldsymbol{z}' - d\boldsymbol{c}_1' \rceil_\gamma), \text{ where} \\ \hat{u} &= \alpha\langle [1 \quad \boldsymbol{b}_2'], \boldsymbol{z}\rangle - \langle [1 \quad \boldsymbol{b}_2'], \boldsymbol{z}'\rangle - (\alpha c_2 + \beta - c_2')d. \end{aligned}$$

The prover then outputs the proof $(d, \boldsymbol{z}, \boldsymbol{z}')$ if $d = d'$ and $\|\boldsymbol{z}[i]\|, \|\boldsymbol{z}'[i]\| \leq 2\sigma_{\mathrm{C}}\sqrt{N}$ (when setting up the check as in [GLP12, BG14], then the test will fail with probability at most 1/2), and the verifier will make the same checks to validate it. The proof size is reduced from k to $k - n$ Gaussian-distributed ring-elements, making the proof size a total of $2(k - n)\log(6\sigma_{\mathrm{C}})$ bits.

4 Protocol: Zero-Knowledge Proof of Correct Shuffle

In this section we present the shuffle protocol for openings of commitments. We construct a public-coin $4 + 3\tau$-move protocol[5] such that the commit-challenge-response stages require the prover to solve a system of linear equations in order to prove a correct shuffle. Our construction extends Neff's construction [Nef01] to the realm of post-quantum assumptions.

The proof of shuffle protocol will use the commitments defined in Section 3. For the shuffle proof to work, the prover $\mathcal{P}$ and verifier $\mathcal{V}$ receive commitments $\{[\![m_i]\!]\}_{i=1}^{\tau}$. $\mathcal{P}$ also receives the set of openings $\{(m_i, \boldsymbol{r}_i)\}_{i=1}^{\tau}$ as well as a permutation $\pi \in S_\tau$. Additionally, both parties also obtain $\{\hat{m}_i\}_{i=1}^{\tau}$.

The goal is to ensure that the following relation R_{Shuffle} holds:

$$R_{\text{Shuffle}} = \left\{ (s, w) \left| \begin{array}{l} s = ([\![m_1]\!], \ldots, [\![m_\tau]\!], \hat{m}_1, \ldots, \hat{m}_\tau, \hat{m}_i \in R_p), \\ w = (\pi, f_1, \ldots, f_\tau, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_\tau), \pi \in S_\tau, \\ \forall i \in [\tau] : \texttt{Open}([\![m_{\pi^{-1}(i)}]\!], \hat{m}_i, \boldsymbol{r}_i, f_i) = 1 \end{array} \right. \right\}.$$

To use the idea of Neff, all $\hat{m}_i$ messages involved have to be invertible. However, this may not be the case for arbitrary ring elements. We start by showing that if $\mathcal{V}$ samples a random ρ in R_p then all $\hat{m}_i - \rho$ will be invertible with high probability:

Proposition 2. *Let $N \geq \delta \geq 1$ be powers of 2, p a prime congruent to $2\delta + 1 \bmod 4\delta$. Then*

$$\Pr_{x_1,\ldots,x_\tau \in R_p}[x_1 - \rho, \ldots, x_\tau - \rho \text{ invertible in } R_p \mid \rho \xleftarrow{\$} R_p] \geq (1 - p^{-N/\delta})^{\delta\cdot\tau}.$$

Plugging in realistic parameters ($p \approx 2^{32}, , N = 2014, \delta = 2, \tau = 1,000,000$) we see that the probability of of all $\hat{m}_i - \rho$ being simultaneously invertible is essentially 1.

Proof. By assumption, R_p factors into δ irreducible factors. The number of invertible elements in each factor of R_p is exactly $p^{N/\delta} - 1$, and hence, the total number of invertible elements in R_p is $(p^{N/\delta} - 1)^\delta$. Dividing by the total number of elements in each factor of R_p and multiplying the probability by itself τ times then gives us that the probability of all $x_i - \rho$ being invertible is bounded below by

$$\left[\left(\frac{p^{N/\delta} - 1}{p^{N/\delta}}\right)^\delta\right]^\tau = (1 - p^{-N/\delta})^{\delta\cdot\tau}.$$

□

The first step for our shuffle protocol will be that $\mathcal{V}$ picks a random appropriate $\rho \xleftarrow{\$} R_p$ and sends ρ to $\mathcal{P}$. $\mathcal{P}$ and $\mathcal{V}$ then locally compute the values $\hat{M}_i, M_i$

[5] This is only a theoretical problem as the protocol is public-coin and can therefore directly be transformed into NIZKs using the Fiat-Shamir transform.

by setting $M_i = m_i - \rho$, $\hat{M}_i = \hat{m}_i - \rho$. The proof, on a high level, then shows that $\prod_i M_i = \prod_i \hat{M}_i$. This is in fact sufficient, as the $m_i, \hat{m}_i$ can be considered as roots of polynomials of degree τ. By subtracting ρ from each such entry and multiplying the results we obtain the evaluation of these implicit polynomials in the point ρ, and if the $\hat{m}_i$ are not a permutation of the m_i then these implicit polynomials will be different. At the same time, the number of points on which both polynomials can agree is upper-bounded as shown in Lemma 2.

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
	$\xleftarrow{\rho}$	$\rho \xleftarrow{\$} R_p \setminus \{\hat{m}_i\}_{i=1}^{\tau}$
$\hat{M}_i = \hat{m}_i - \rho$		$\hat{M}_i = \hat{m}_i - \rho$
$M_i = m_i - \rho$		$[\![M_i]\!] = [m_i] - \rho$
$\theta_i \xleftarrow{\$} R_p, \forall i \in [\tau - 1]$		
Compute $[\![D_i]\!]$ as in Eq. (1), i.e.		
$[\![D_1]\!] = [\![\theta_1 \hat{M}_1]\!], [\![D_\tau]\!] = [\![\theta_{\tau-1} M_\tau]\!]$,		
$[\![D_i]\!] = [\![\theta_{i-1} M_i + \theta_i \hat{M}_i]\!]$ for $i \in [\tau - 1] \setminus \{1\}$	$\xrightarrow{\{[\![D_i]\!]\}_{i=1}^{\tau}}$	
	$\xleftarrow{\beta}$	$\beta \xleftarrow{\$} R_p$
Compute $s_i, \forall i \in [\tau - 1]$ as in (3).	$\xrightarrow{\{s_i\}_{i=1}^{\tau-1}}$	
		Use Π_{Lin} to prove that
		(1) $\beta [\![M_1]\!] + s_1 \hat{M}_1 = [\![D_1]\!]$
		(2) $\forall i \in [\tau - 1] \setminus \{1\}: \; s_{i-1} [\![M_i]\!] + s_i \hat{M}_i = [\![D_i]\!]$
		(3) $s_{\tau-1} [\![M_\tau]\!] + (-1)^\tau \beta \hat{M}_\tau = [\![D_\tau]\!]$
		i.e. all equations from (2)
		return accept iff all instances of Π_{Lin} are accepting

Fig. 2: The public-coin zero-knowledge protocol of correct shuffle Π_{Shuffle}.

Our public-coin zero-knowledge protocol proves this identity of evaluations of these two polynomials by showing that a particular set of linear relations (2) is satisfied (we will show later how it is related to the aforementioned product of M_i and $\hat{M}_i$).

As a first step, $\mathcal{P}$ draws $\theta_i \xleftarrow{\$} R_p$ uniformly at random for each $i \in \{1, \dots, \tau\}$, and computes the commitments

$$\begin{aligned} [\![D_1]\!] &= [\![\theta_1 \hat{M}_1]\!] \\ \forall j \in \{2, \dots, \tau - 1\}: \quad [\![D_j]\!] &= [\![\theta_{j-1} M_j + \theta_j \hat{M}_j]\!] \\ [\![D_\tau]\!] &= [\![\theta_{\tau-1} M_\tau]\!]. \end{aligned} \tag{1}$$

$\mathcal{P}$ then sends these commitments $\{[\![D_i]\!]\}_{i=1}^{\tau}$ to the verifier[6] $\mathcal{V}$, which in turn chooses a challenge $\beta \in R_p$, whereupon $\mathcal{P}$ computes $s_i \in R_q$ such that the following equations are satisfied:

[6] $\mathcal{P}$ does not show that these commitments are well-formed, this will not be necessary.

$$\begin{aligned} \beta M_1 + s_1 \hat{M}_1 &= \theta_1 \hat{M}_1 \\ \forall j \in \{2, \ldots, \tau-1\} : \; s_{j-1} M_j + s_j \hat{M}_j &= \theta_{j-1} M_j + \theta_j \hat{M}_j \\ s_{\tau-1} M_\tau + (-1)^\tau \beta \hat{M}_\tau &= \theta_{\tau-1} M_\tau . \end{aligned} \tag{2}$$

To verify the relations, $\mathcal{P}$ uses the protocol Π_{Lin} from Section 3 to prove that the content of each commitment $[\![D_i]\!]$ is such that D_i, M_i and $\hat{M}_i$ satisfies the equations (2). The protocol ends when $\mathcal{V}$ has verified all the τ linear equations in (2). In order to compute the s_i values, we can use the following fact:

Lemma 4. *Choosing*

$$s_j = (-1)^j \cdot \beta \prod_{i=1}^{j} \frac{M_i}{\hat{M}_i} + \theta_j \tag{3}$$

for all $j \in 1, \ldots, \tau-1$ *yields a valid assignment for Equation* (2).

Proof. The correctness of this choice follows directly by considering all three cases: For the first case, we have that

$$\begin{aligned} \beta M_1 + s_1 \hat{M}_1 &= \beta M_1 + (-\beta M_1/\hat{M}_1 + \theta_1)\hat{M}_1 \\ &= \theta_1 \hat{M}_1 . \end{aligned}$$

In the second case, it holds that

$$\begin{aligned} s_{j-1} M_j + s_j \hat{M}_j &= ((-1)^{j-1}\beta \prod_{i=1}^{j-1} M_i/\hat{M}_i + \theta_{j-1}) M_j + ((-1)^j \beta \prod_{i=1}^{j} M_i/\hat{M}_i + \theta_1)\hat{M}_j \\ &= \theta_{j-1} M_j + \theta_j \hat{M}_j \end{aligned}$$

where the β-terms cancel. For the third case, since $\dfrac{M_1 \cdots M_\tau}{\hat{M}_1 \cdots \hat{M}_{\tau-1}} = \hat{M}_\tau$ so

$$\begin{aligned} s_{\tau-1} M_\tau + (-1)^\tau \beta \hat{M}_\tau &= ((-1)^{\tau-1}\beta \prod_{i=1}^{\tau-1} M_i/\hat{M}_i + \theta_{\tau-1}) M_\tau + (-1)^\tau \beta \hat{M}_\tau \\ &= \theta_{\tau-1} M_\tau . \end{aligned}$$

□

From Lemma 4 it is clear that the protocol is indeed complete. Interestingly, this choice of s_j also makes these values appear random: each s_j is formed by adding a fixed term to a uniformly random private value θ_j. This will be crucial to show the zero-knowledge property. For the soundness, we get the following:

Lemma 5. *Assume that the commitment scheme is binding and that* Π_{Lin} *is a sound proof of knowledge for the relation* R_{Lin} *except with probability* t. *Then the protocol in Figure 2 is a sound proof of knowledge for the relation* R_{Shuffle} *except with probability* $\epsilon \leq \frac{\tau^\delta+1}{|R_p|} + 4\tau t$.

Proof. To prove the statement, we will construct a PPT algorithm $\mathcal{E}$ called *extractor* which interacts with $\mathcal{P}^*$ in a black-box manner and which will output a witness w for a statement s such that $(s, w) \in R_{\text{Shuffle}}$ given that $\mathcal{P}^*$ wins for a given s with more than the stated probability. The expected runtime of $\mathcal{E}$ is $poly(\tau, t)/\epsilon$. The extractor algorithm will proceed as follows:

1. Construct sub-extractors $\mathcal{E}_i$ which for each $i \in [\tau]$ do the following:
   (a) Run instances with an arbitrary randomness tape for $\mathcal{P}^*$ as well as arbitrary challenges until an accepting transcript is found.
   (b) Upon finding an accepting transcript, rewind $\mathcal{P}^*$ until after the first message in the ith instance of Π_{Lin} was sent. Then probe for a second challenge for the ith proof that leads to an accepting transcript[7].
2. Subtract ρ_i from all M_i where ρ_i is the value used by the extractor $\mathcal{E}_i$. Then for each $[\![m_i]\!]$ let the opening be $(m_i; \boldsymbol{r}_i, f_i)$. If the m_i are indeed a permutation of the $\hat{m}_i$ then output the respective $w = (\pi, f_1, \ldots, f_\tau, \boldsymbol{r}, \ldots, \boldsymbol{r}_\tau)$.

We will first argue why the above algorithm is expected polynomial-time: The runtime of each $\mathcal{E}_i$ is expected polynomial time by a standard heavy-row argument, as we only need to rewind on a specific instance only. Applying the heavy-row argument is possible as the success probability of $\mathcal{P}^*$ is above $4t$ (we lose a factor of 4 in the heavy-row argument). We now argue why also Step 2 is polynomial-time i.e. that $\mathcal{E}$ outputs a witness.

Let us assume we would create two transcripts for an identical ρ but differing β, β' where we rewind $\mathcal{P}^*$. We would then obtain different s_i, s_i' such that all the equations are proven by $\mathcal{P}^*$ with Π_{Lin}. From the soundness of Π_{Lin} we obtain:

1. $\beta M_1 + s_1 \hat{M}_1 = D_1$ and $\beta' M_1 + s_1' \hat{M}_1 = D_1$,
2. $\forall i \in [\tau - 1] \setminus \{1\} :\ s_{i-1} M_i + s_i \hat{M}_i = D_i$ and $s_{i-1}' M_i + s_i' \hat{M}_i = D_i$,
3. $s_{\tau-1} M_\tau + (-1)^\tau \beta \hat{M}_\tau = D_\tau$ and $s_{\tau-1}' M_\tau + (-1)^\tau \beta' \hat{M}_\tau = D_\tau$.

Subtracting the equations with identical D_i on the right-hand side yields the following system of τ linear equations:

$$\underbrace{\begin{bmatrix} M_1 & \hat{M}_1 & 0 & \ldots & 0 & 0 \\ 0 & M_2 & \hat{M}_2 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & M_{\tau-1} & \hat{M}_{\tau-1} \\ (-1)^\tau \hat{M}_\tau & 0 & 0 & \ldots & 0 & M_\tau \end{bmatrix}}_{\boldsymbol{M}} \underbrace{\begin{bmatrix} \beta - \beta' \\ s_1 - s_1' \\ \vdots \\ s_{\tau-2} - s_{\tau-2}' \\ s_{\tau-1} - s_{\tau-1}' \end{bmatrix}}_{\boldsymbol{c}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}. \tag{4}$$

We can directly see from the above that all $M_i, s_i - s_i'$ must be non-zero and that therefore $s_i \neq s_i'$: From the last equation and the assumption that $\beta \neq \beta'$

[7] One would additionally in parallel run a process that aborts $\mathcal{E}_i$ with very small probability, see e.g. [BBC+18, Lemma 3]. We leave this out for the sake of simplicity.

we know that $M_\tau \times (s_{\tau-1} - s'_{\tau-1}) \neq 0$ as $\hat{M}_\tau$ is invertible. From the second-to-last equation and due to the invertibility of $\hat{M}_{\tau-1}$ the same holds for $M_{\tau-1}$ and $s_{\tau-2} - s'_{\tau-2}$. By induction, this applies to all values.

By a standard rule from linear algebra, we know that if $\det(\boldsymbol{M}) \neq 0$ then $\mathbf{0}$ is the only element in the kernel of $\boldsymbol{M}$, while $\boldsymbol{c} \neq \mathbf{0}$. Therefore, $\det(\boldsymbol{M}) = 0$. We explicitly compute $\det(\boldsymbol{M})$ using the Laplace expansion obtained by removing the first column (and then all rows successively). As most of the cofactors are 0, we obtain $0 = \det(\boldsymbol{M}) = M_1 \cdot \det(\boldsymbol{M}_1) + (-1)^{2\tau+1} \cdot \det(\boldsymbol{M}_2)$ where

$$\boldsymbol{M}_1 = \begin{bmatrix} M_2 & \hat{M}_2 & \dots & 0 & 0 \\ 0 & M_3 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & M_{\tau-1} & \hat{M}_{\tau-1} \\ 0 & 0 & \dots & 0 & M_\tau \end{bmatrix}, \qquad \boldsymbol{M}_2 = \begin{bmatrix} \hat{M}_1 & 0 & \dots & 0 & 0 \\ M_2 & \hat{M}_2 & \dots & 0 & 0 \\ 0 & M_3 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & M_{\tau-1} & \hat{M}_{\tau-1} \end{bmatrix}.$$

Clearly $\det(\boldsymbol{M}_1) = M_2 \cdots M_\tau$ and $\det(\boldsymbol{M}_2) = \hat{M}_1 \cdots \hat{M}_{\tau-1}$ and therefore $\det(\boldsymbol{M}) = \prod_{i=1}^{\tau} M_i - \prod_{i=1}^{\tau} \hat{M}_j$ and $\prod_{i=1}^{\tau} M_i = \prod_{i=1}^{\tau} \hat{M}_j \neq 0$. Define the two polynomials

$$g(X) = \prod\nolimits_{i=1}^{\tau} (m_i - X), \text{ and } \hat{g}(X) = \prod\nolimits_{i=1}^{\tau} (\hat{m}_i - X),$$

that is, $g(X)$ and $\hat{g}(X)$ are the polynomials which have m_i and $\hat{m}_i$ as their roots, respectively. If there exists no permutation such that $\hat{m}_i = m_{\pi(i)}, \forall i = 1, \dots, \tau$, then $g(\rho) = \hat{g}(\rho)$ (i.e. $\det(\boldsymbol{M}) = 0$) for at most τ^δ choices of ρ according to Lemma 2. Due to the lower-bound on the success probability of $\mathcal{P}^*$ there must exist accepting transcripts for more than τ^δ choices of ρ with more than one accepting choice of β where all instances of Π_{Lin} prove correct. Then, by the above argument and because the commitment scheme is binding, it must hold that the values extracted by $\mathcal{E}$ indeed are a permutation of the $\hat{m}_i$. □

From Lemmas 4 and 5 we get the following theorem:

Theorem 1. *Assume that* $(\mathtt{KeyGen}_\mathrm{C}, \mathtt{Com}, \mathtt{Open})$ *is a secure commitment scheme with* Π_{Lin} *as a HVZK Proof of Knowledge of the relation* $\mathcal{R}_{\text{Lin}}$ *with soundness error* t. *Then the protocol* Π_{Shuffle} *is an HVZK Proof of Knowledge for the relation* $\mathcal{R}_{\text{Shuffle}}$ *with soundness error* $(\tau^\delta + 1)/|R_p| + 4\tau t$.

The proof of completeness and HVZK can be found in Appendix A.

5 Applications to Electronic Voting

We now construct an e-voting protocol by combining the shuffle protocol from Section 4 with a verifiable encryption scheme and a return code mechanism.

Towards this end, consider the shuffled openings of commitments as the outcome of the election, meaning that each commitment will contain a vote.

Commitments are not sufficient for a voting system, and we also need encryptions of the actual ballots and these must be tied to the commitments, so that the shuffling server can open the commitments without anyone else being able to. We use a version of the verifiable encryption scheme by Lyubashevsky and Neven [LN17] to verifiably encrypt openings under a public key that belongs to the shuffle server. We also reuse the verifiable encryption to get a system for return codes. The return code computation is done in two stages, where the first stage is done on the voter's computer, and the second stage is done by an infrastructure player. The voter's computer commits to its result and verifiably encrypts an opening of that commitment for the infrastructure player. Then it proves that the commitment contains the correct value.

We will now describe the verifiable encryption scheme that we use as well as the return code mechanism in more detail, before explaining how to construct the full e-voting protocol.

5.1 Verifiable Encryption

In a *verifiable encryption scheme*, anyone can verify that the encrypted plaintext has certain properties. We use a version of [LN17] where we use a generalization of the [LPR13, BGV12] encryption system. The reason is that in [LN17] the public key only consists of single polynomials of degree N, requiring that the plaintext vector must also be a multiple of N - which might not always be the case as in our setting.

In our setting, the goal is to show that the plaintext is a value $\boldsymbol{\mu} \in D^{\kappa}_{\beta_\infty}$ such that

$$\boldsymbol{T}\boldsymbol{\mu} = \boldsymbol{u} \bmod p, \tag{5}$$

for some fixed $\boldsymbol{T}, \boldsymbol{u}$ and where $\boldsymbol{T} \in R_p^{\lambda \times \kappa}$. Using the construction of [LN17], one can show a weaker version of the statement, namely that decryption yields a small $\bar{\boldsymbol{\mu}}$ and $\bar{c} \in \overline{\mathcal{C}}$ over R_p such that

$$\boldsymbol{T}\bar{\boldsymbol{\mu}} = \bar{c}\boldsymbol{u} \bmod p. \tag{6}$$

We will see that this will be sufficient for our voting scheme[8].

The [LN17] verifiable encryption scheme consists of 4 algorithms: Key generation $\texttt{KeyGen}_V$, encryption `Enc`, verification `Ver` and decryption `Dec`. We will first describe the underlying non-verifiable encryption scheme and then explain how it is made verifiable.

The encryption, verification and decryption algorithms are described in Figures 3, 4 and 5 respectively. Here, encryption follows [LPR13, BGV12] but additionally computes a NIZK that the plaintext is a valid preimage of Equation 5 and also bounded. `Ver` validates the NIZK, while `Dec` decrypts to a short plaintext that is valid under Equation 6.

[8] Recently, [ALS20] showed a more efficient HVZKPoK for the respective relation. Unfortunately, their proof cannot guarantee that $\bar{c}$ is invertible, which is crucial for the verifiability of the encryption scheme. Their optimization can therefore not be applied in our setting.

To generate a public key $(\boldsymbol{A}, \boldsymbol{t})$ for the verifiable encryption scheme one samples $\boldsymbol{A} \leftarrow R_q^{\ell \times \ell}$ uniformly at random as well as $\boldsymbol{s}_1, \boldsymbol{s}_2 \leftarrow D_1^{\ell}$, setting $\boldsymbol{t} \leftarrow \boldsymbol{A}\boldsymbol{s}_1 + \boldsymbol{s}_2$ and outputting $(\boldsymbol{A}, \boldsymbol{t})$ as public key as well as $\boldsymbol{s_1}$ as private key. To encrypt a single message $\mu \in D_{\beta_\infty}$, first sample $\boldsymbol{r}, \boldsymbol{e} \leftarrow D_1^{\ell}, e' \leftarrow D_1$ and then compute

$$\begin{bmatrix} \boldsymbol{v} \\ w \end{bmatrix} = \begin{bmatrix} p(\boldsymbol{A}^\top \boldsymbol{r} + \boldsymbol{e}) \bmod q \\ p(\langle \boldsymbol{t}, \boldsymbol{r} \rangle + e') + \mu \bmod q \end{bmatrix}.$$

To decrypt a ciphertext $(\boldsymbol{v}, w) \in R_q^{\ell} \times R_q$ compute

$$w - \langle \boldsymbol{v}, \boldsymbol{s}_1 \rangle \bmod q \bmod p = (p(\langle \boldsymbol{r}, \boldsymbol{s}_2 \rangle + e' - \langle \boldsymbol{e}, \boldsymbol{s}_1 \rangle) + \mu \bmod q) \bmod p = \mu,$$

where the last equality holds if $\|p(\langle \boldsymbol{r}, \boldsymbol{s}_2 \rangle + e' - \langle \boldsymbol{e}, \boldsymbol{s}_1 \rangle + \mu)\|_\infty < q/2$.

To encrypt a vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_\kappa) \in D_{\beta_\infty}^{\kappa}$ we can abbreviate our equations in matrix notation in the following way:

$$\begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell \cdot \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa \times (\ell \cdot \kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \end{bmatrix} \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{e} \\ \boldsymbol{e}' \\ \boldsymbol{\mu} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix} \bmod q, \tag{7}$$

for $\boldsymbol{r}, \boldsymbol{e} \xleftarrow{\$} D_1^{\ell \cdot \kappa}, \boldsymbol{e}' \xleftarrow{\$} D_1^{\kappa}$ and where $\boldsymbol{v} \in R_q^{\ell \cdot \kappa}, \boldsymbol{w} \in R_q^{\kappa}$. For decryption we get:

$$\boldsymbol{\mu}[i] = \boldsymbol{w}[i] - \langle \boldsymbol{s}_1, \hat{\boldsymbol{v}}_i \rangle \bmod q \bmod p,$$

where $\hat{\boldsymbol{v}}_i = [\boldsymbol{v}[(i-1) \cdot \ell + 1], \boldsymbol{v}[(i-1) \cdot \ell + 2], \ldots, \boldsymbol{v}[i \cdot \ell]]^\top$ which follows from the definition of the tensor product. For correctness, the aforementioned decryption bound generalizes to

$$\|p(\langle \hat{\boldsymbol{r}}_i, \boldsymbol{s}_2 \rangle + \boldsymbol{e}'[i] - \langle \hat{\mathbf{e}}, \boldsymbol{s}_1 \rangle) + \boldsymbol{\mu}[i]\|_\infty < q/2,$$

where $\hat{\boldsymbol{r}}_i, \hat{\boldsymbol{e}}_i$ are analogously defined to $\hat{\boldsymbol{v}}_i$. Using standard bounds on ∞-norms of products of elements in R_p, this directly translates into the requirement that $q > 2p(2\ell \cdot N^2 \cdot {\beta_\infty}^2 + N + 1)$.

To make this verifiable, we will use a NIZK which shows for a ciphertext $\boldsymbol{v}, \boldsymbol{w}$ that the sender knows $\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{e}', \boldsymbol{\mu}$ that are bounded such that

- $\boldsymbol{r}, \boldsymbol{e}, \boldsymbol{e}', \boldsymbol{\mu}$ are a preimage of $\boldsymbol{v}, \boldsymbol{w}$ modulo q as in Equation 7.
- $\boldsymbol{\mu}$ fulfills equation (5) modulo p.

To prove both relations simultaneously we use a standard lattice-based zero-knowledge proof for the specific relation, and by using the Fiat-Shamir transform [FS87] this then becomes non-interactive. As mentioned before, the output of `Dec` will be a witness for Equation 6, i.e. it will also contain the additional factor $\bar{c}$.

We now argue that our modified scheme [LN17] is still secure.

First of all, the encryption scheme (as the authors of [BGV12] show) can be generalized to work with the generalized M-LWE assumption[9]. As the actual

[9] M-LWE generalizes the Ring-LWE assumption [LS15] and is a more conservative security assumption for the same dimensions of the matrix.

matrix dimensions for the encryption scheme do not change between our instance of the verifiable encryption scheme and [LN17] the same security proof still applies with respect to privacy.

The verifiability and thus decryption of the above construction directly follows from the original proof, as neither of the conditions of [LN17, Lemma 3.1] are altered by changing the matrix structure. Furthermore, as we will choose $\ell = 2$ in our setting even this R_q of smaller dimension than in the original work has a large enough challenge space necessary for the (non-)interactive proofs to be sound. We will therefore be able to basically rely on the same security analysis as [LN17] and can essentially re-use their parameters (with some slightly increased p, q).

There are multiple parameter restrictions in [LN17] in order to achieve security. These also apply to our setting:

1. The underlying encryption scheme must safely be able to encrypt and decrypt messages from R_p^κ. For this, we obviously need that message and noise, upon decryption, do not "overflow" $\bmod q$ while the noise at the same time must be large enough such that the underlying MLWE-problem is hard. For concrete parameters, the latter can be established by e.g. using the LWE Estimator [APS15]. For correctness of the decryption alone, we require that the decryption of a correct encryption must yield[10] a value $< q/2$. This also means that the decryption algorithm will always terminate for $\bar{c} = 1$ in case the encryptor is honest.
2. The NIZK requires "quasi-unique responses", which (as the authors of [LN17] argue) it will have with overwhelming probability over the choice of $\boldsymbol{A}$ as long as $24\sigma_{\mathrm{E}}^2 < q$.

Encrypting openings of commitments. We want to make sure that the voter actually knows his vote, and that the commitment and the opening of the commitment are well-formed. We also want to ensure that the ciphertext actually contains a valid opening of the commitment. This can be achieved if the voter creates a proof that the underlying plaintext is an opening of the commitment. Then the ballot box can ensure that the shuffle server will be able to decrypt the vote and use it in the shuffle protocol. Note that the voter may send a well-formed but invalid vote, but then the shuffle server can publicly discard that vote later, and everyone can check that the vote indeed was invalid.

Recall that the commitment is of the form

$$\mathtt{Com}(m; \boldsymbol{r}_m) = \begin{bmatrix} \boldsymbol{c}_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{b}_2 \end{bmatrix} \cdot \boldsymbol{r}_m + \begin{bmatrix} \boldsymbol{0} \\ m \end{bmatrix}.$$

The value $\boldsymbol{c}_1$ serves to bind the committer to a single choice of $\boldsymbol{r}_m$, while c_2 hides the actual message using the unique $\boldsymbol{r}_m$. Fixing $\boldsymbol{r}_m$ fixes m uniquely, and m can indeed be recovered using $\boldsymbol{r}_m$ only. The idea is to use the verifiable

[10] This translates into the requirement that $q > 2p(2\ell \cdot N^2 \cdot {\beta_\infty}^2 + N + 1)$.

```
**Input:** Public key $\mathtt{pk} = (\boldsymbol{A}, \boldsymbol{t}, p, q)$, pair $(\boldsymbol{T}, \boldsymbol{u}), \boldsymbol{\mu} \in D_\beta^\kappa$ such that
$\boldsymbol{T}\boldsymbol{\mu} = \boldsymbol{u}$, hash function $H : \{0,1\}^* \to \mathcal{C}$,
$\sigma_\mathrm{E} = 11 \cdot \max_{c \in \mathcal{C}} \|c\| \cdot \sqrt{\kappa N (3 + \beta)}$
**Output:** ciphertext $(\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}) \in R_q^{\ell \cdot \kappa} \times R_q^\kappa \times \mathcal{C} \times R^{(2\ell+2)\kappa}$
1 : $\boldsymbol{r}, \boldsymbol{e} \xleftarrow{\$} D_1^{\ell \cdot \kappa}, \boldsymbol{e}' \xleftarrow{\$} D_1^\kappa$
2 : $\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell \cdot \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa \times (\ell \cdot \kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \end{bmatrix} \begin{bmatrix} \boldsymbol{r} \ \boldsymbol{e} \ \boldsymbol{e}' \ \boldsymbol{\mu} \end{bmatrix}^\top$
3 : $\boldsymbol{y} \leftarrow \begin{bmatrix} \boldsymbol{y_r} \ \boldsymbol{y_e} \ \boldsymbol{y_{e'}} \ \boldsymbol{y_\mu} \end{bmatrix}^\top \xleftarrow{\$} D_{R^{(2\ell+2)\kappa}, \boldsymbol{0}, \sigma_\mathrm{E}}$
4 : $\boldsymbol{Y} \leftarrow \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell \cdot \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa \times (\ell \cdot \kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda \times (\ell \cdot \kappa)} & \boldsymbol{0}^{\lambda \times (\ell \cdot \kappa)} & \boldsymbol{0}^{\lambda \times \kappa} & \boldsymbol{T} \end{bmatrix} \begin{bmatrix} \boldsymbol{y_r} \ \boldsymbol{y_e} \ \boldsymbol{y_{e'}} \ \boldsymbol{y_\mu} \end{bmatrix}^\top \begin{matrix} \bmod q \\ \bmod q \\ \bmod p \end{matrix}$
5 : $c \leftarrow H\left( \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell \cdot \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa \times (\ell \cdot \kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda \times (\ell \cdot \kappa)} & \boldsymbol{0}^{\lambda \times (\ell \cdot \kappa)} & \boldsymbol{0}^{\lambda \times \kappa} & \boldsymbol{T} \end{bmatrix}, \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \\ \boldsymbol{u} \end{bmatrix}, \boldsymbol{Y} \right)$
6 : $\boldsymbol{s} \leftarrow \begin{bmatrix} \boldsymbol{r} \ \boldsymbol{e} \ \boldsymbol{e}' \ \boldsymbol{\mu} \end{bmatrix}^\top c$
7 : $\boldsymbol{z} \leftarrow \boldsymbol{s} + \boldsymbol{y}$
8 : With probability $1 - \min\left(1, \frac{D_{R^{(2\ell+2)\kappa}, \boldsymbol{0}, \sigma_\mathrm{E}}(\boldsymbol{z})}{3 \cdot D_{R^{(2\ell+2)\kappa}, \boldsymbol{s}, \sigma_\mathrm{E}}(\boldsymbol{z})}\right)$ goto 3
9 : **if** $\|\boldsymbol{z}\|_\infty \geq 6\sigma_\mathrm{E}$ goto 3, **else return** $\boldsymbol{e} = (\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z})$
```

Fig. 3: The verifiable encryption algorithm `Enc`.

```
**Input:** Secret key $\mathtt{sk} = (\boldsymbol{s}_1)$, pair $x = (\boldsymbol{T}, \boldsymbol{u})$,
 ciphertext $t = (\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}), C = \max_{c \in \mathcal{C}} ||c||_\infty$
1 : if $\mathtt{Ver}(t, x, \mathtt{pk}) = 1$ then
2 :   while
3 :     $c' \xleftarrow{\$} \mathcal{C}$
4 :     $\overline{c} \leftarrow c - c'$
5 :     $\overline{\boldsymbol{m}}[i] \leftarrow (\boldsymbol{w} - \langle \boldsymbol{s}_1, \boldsymbol{v}_i \rangle)\overline{c} \bmod q$ for all $i \in [\kappa]$
6 :     if $||\overline{\boldsymbol{m}}||_\infty \leq q/2C$ and $||\overline{\boldsymbol{m}} \bmod p||_\infty < 12\sigma_E$ then
7 :       return $(\overline{\boldsymbol{m}} \bmod p, \overline{c})$
```

Fig. 4: Algorithm `Dec` for decryption of a ciphertext.

```
**Input:** ciphertext $t = (\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}) \in R_q^{\ell \cdot \kappa} \times R_q^\kappa \times R_q \times R^{(2\ell+2)\kappa}$, language element $x = (\boldsymbol{T}, \boldsymbol{u})$,
 public key $\mathtt{pk} = (\boldsymbol{A}, \boldsymbol{t}, p, q)$
1 : if $\|\boldsymbol{z}\|_\infty > 6 \cdot \sigma_\mathrm{E}$ then return 0
2 : $\boldsymbol{Z} \leftarrow \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell \cdot \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa \times (\ell \cdot \kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda \times (\ell \cdot \kappa)} & \boldsymbol{0}^{\lambda \times (\ell \cdot \kappa)} & \boldsymbol{0}^{\lambda \times \kappa} & \boldsymbol{T} \end{bmatrix} \boldsymbol{z} - c \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \\ \boldsymbol{u} \end{bmatrix} \begin{matrix} \bmod q \\ \bmod q \\ \bmod p \end{matrix}$
3 : if $c \neq \mathrm{H}\left( \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell \cdot \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} & \boldsymbol{0}^{(\ell \cdot \kappa) \times \kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa \times (\ell \cdot \kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda \times (\ell \cdot \kappa)} & \boldsymbol{0}^{\lambda \times (\ell \cdot \kappa)} & \boldsymbol{0}^{\lambda \times \kappa} & \boldsymbol{T} \end{bmatrix}, \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \\ \boldsymbol{u} \end{bmatrix}, \boldsymbol{Z} \right)$ then return 0
4 : return 1
```

Fig. 5: Algorithm `Ver` for verification of a ciphertext.

encryption scheme to encrypt the opening $\boldsymbol{r}_m$, and prove that the voter knows a witness for the relation $\boldsymbol{c}_1 = \boldsymbol{B_1}\boldsymbol{r}_m \bmod p$ where $\boldsymbol{r}_m$ is bounded. Any such randomness could then be used to uniquely open the commitment.

To encrypt the opening $\boldsymbol{r}_m$ verifiably, Step 4 in Figure 3 is now the system

$$\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \\ \boldsymbol{c}_1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}^\top \otimes (p\boldsymbol{I}_\kappa) & p\boldsymbol{I}_{\ell\cdot\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} & \boldsymbol{0}^{(\ell\cdot\kappa)\times\kappa} \\ \boldsymbol{t}^\top \otimes (p\boldsymbol{I}_\kappa) & \boldsymbol{0}^{\kappa\times(\ell\cdot\kappa)} & p\boldsymbol{I}_\kappa & \boldsymbol{I}_\kappa \\ \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times(\ell\cdot\kappa)} & \boldsymbol{0}^{\lambda\times\kappa} & \boldsymbol{B}_1 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{e} \\ \boldsymbol{e}' \\ \boldsymbol{r}_m \end{bmatrix}.$$

Note that the encryption of the opening $\boldsymbol{r}_m$ now contains two parts: $\boldsymbol{v}$ and $\boldsymbol{w}$ correspond to the ciphertext of the encryption while $\boldsymbol{c}_1$ corresponds to the verification of the opening of the commitment.

5.2 Return Codes

In the case of a malicious computer, we need to make sure that the voter can detect if the encrypted vote being sent to the ballot box is not an encryption of the correct ballot. We achieve this by giving the voter a pre-computed table of return codes which he can use for verification. The return codes are generated per voter, using a voter-unique blinding-key and a system-wide PRF-key.

A commitment to the blinding key is made public. The computer gets the blinding-key and must create a pre-code by blinding the ballot with the blinding-key. The computer also generates commitments to the ballot and the pre-code, along with a proof that the pre-code has been generated correctly. Anyone with an opening of the pre-code commitment and the PRF-key can now generate the correct return code without learning anything about the ballot.

Defining the Return Code. Assume that the voters have ω different options in the election. Let $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_\omega \in R_p$ be ballots, let $a_j \in R_p$ be a blinding-key for a voter V_j and let $\mathtt{PRF}_k : \{0,1\}^* \times R_p \to \{0,1\}^n$ be a pseudo-random function, instantiated with a PRF-key k, from pairs of binary strings and elements from R_p to the set of binary strings of length n. The *pre-code* $\hat{r}_{ij}$ corresponding to the ballot $\hat{v}_i$ is $\hat{r}_{ij} = a_j + \hat{v}_i \mod p$. The *return code* r_{ij} corresponding to the ballot $\hat{v}_i$ is $r_{ij} = \mathtt{PRF}_k(V_j, \hat{r}_{ij})$.

Let $\boldsymbol{c}_{a_j}$, $\boldsymbol{c}_j$ and $\boldsymbol{c}_{\hat{r}_j}$ be commitments to the blinding key a_j, the ballot $v_j \in \{\hat{v}_1, \ldots, \hat{v}_\omega\}$ and the pre-code $\hat{r}_j = a_j + v_j \mod p$. It is now clear that we can prove that a given $\hat{r}_j$ value has been correctly computed by giving a proof of linearity that $a_j + v_j = \hat{r}_j$. This can be done either by adding the commitments $\boldsymbol{c}_{a_j}$ and $\boldsymbol{c}_j$ together directly to get a commitment $\boldsymbol{c}_{a_j+v_j}$ with larger randomness (if the choice of parameters allows for the sum of the randomness to be a valid opening of the commitment) and then prove the equality of the committed messages, or to extend the proof of linearity to handle three terms. Our return code construction is now straight-forward.

A commitment $\boldsymbol{c}_{a_j}$ to voter V_j's voter-unique blinding key a_j is public. The voter V_j's *computer* will get the voter-unique blinding-key a_j together with the randomness used to create $\boldsymbol{c}_{a_j}$. It has already created a commitment $\boldsymbol{c}_j$ to the ballot v_j. It will compute the pre-code $\hat{r}_j = a_j + v_j$, a commitment $\boldsymbol{c}_{\hat{r}_j}$ to $\hat{r}_j$ and a proof $\Pi_{\hat{r}_j}$ of knowledge of the opening of that sum. Finally, it will verifiably encrypt as $\boldsymbol{e}_{\hat{r}_j}$ the opening of $\boldsymbol{c}_{\hat{r}_j}$ with the return code generator's public key.

The *return code generator* receives V_j, $\boldsymbol{c}_{\hat{r}_j}$, $\boldsymbol{c}_j$, $\boldsymbol{e}_{\hat{r}_j}$ and $\Pi_{\hat{r}_j}$. It verifies the proof and the encryption, and then decrypts the ciphertext to get $\hat{r}_j$. It computes the return code as $r_j = \mathtt{PRF}_k(V_j, \hat{r}_j)$.

Note that if a voter re-votes (such as when exposed to coercion), the return code generator would be able to learn something about the ballots involved. The return code mechanism can be extended for re-voting using higher degree polynomials to hide the vote.

5.3 The Voting Scheme

We get our e-voting protocol by combining the shuffle protocol with the verifiable encryption scheme and the return code construction. A complete description of this protocol can be found in Figure 6 and in Appendix B. All communication happens over secure channels. We discuss the *privacy*, *integrity* and *coercion resistance* of the voting scheme in detail in Appendix B.

Registration phase. The only thing that happens in this phase is key generation. Every player generates their own key material and publishes the public keys and any other commitments.

The voter's computer, the return code generator and a *trusted printer* then use a multi-party computation protocol[11] to compute the ballot-return code pairs for the voter, such that only the trusted printer learns the pairs. The trusted printer then sends these pairs to the voter through a secure channel. We emphasize that for many voters, the registration phase likely requires significant computational resources for the return code generator and the trusted printer. In practice, the voter's computer will usually play a minor role in this key generation.

Casting a ballot. The voter begins the ballot casting by giving the ballot v_j to the voter's computer.

The voter's computer has the per-voter secret key material, and gets the ballot v_j to be cast from the voter. It computes the pre-code $\hat{r}$ and generates commitments $\boldsymbol{c}_j$ and $\boldsymbol{c}_{\hat{r}}$ to the ballot and the pre-code, respectively. It creates a proof $\Pi_{\hat{r}}$ that the pre-code has been correctly computed (with respect to the commitments). It creates a verifiable encryption $\boldsymbol{e}_j$ of an opening of $\boldsymbol{c}_j$ to v_j under the shuffle server's public key, and a verifiable encryption $\boldsymbol{e}_{\hat{r}}$ of an opening of $\boldsymbol{c}_{\hat{r}}$ under the return code generator's public key. It then signs all of

[11] Since this happens before the election, speed is no longer essential. Even so, for the computations involved here, ordinary MPC is sufficiently practical. In a practical deployment, the voter's computer is unlikely to be part of this computation. It would instead be delegated to a set of trusted key generation players.

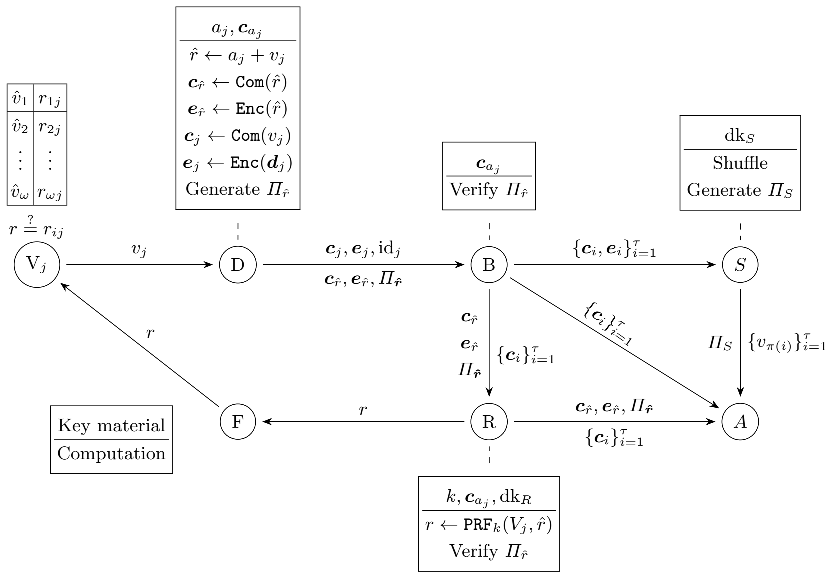
$\hat{v}_1$ r_{1j}
$\hat{v}_2$ r_{2j}
$\hat{v}_\omega$ $r_{\omega j}$
$r \overset{?}{=} r_{ij}$
$a_j, \boldsymbol{c}_{a_j}$
$\hat{r} \leftarrow a_j + v_j$
$\boldsymbol{c}_{\hat{r}} \leftarrow \texttt{Com}(\hat{r})$
$\boldsymbol{e}_{\hat{r}} \leftarrow \texttt{Enc}(\hat{r})$
$\boldsymbol{c}_j \leftarrow \texttt{Com}(v_j)$
$\boldsymbol{e}_j \leftarrow \texttt{Enc}(\boldsymbol{d}_j)$
Generate $\Pi_{\hat{r}}$
$\boldsymbol{c}_{a_j}$
Verify $\Pi_{\hat{r}}$
dk_S
Shuffle
Generate Π_S
V_j
v_j
D
$\boldsymbol{c}_j, \boldsymbol{e}_j, \mathrm{id}_j$
$\boldsymbol{c}_{\hat{r}}, \boldsymbol{e}_{\hat{r}}, \Pi_{\hat{r}}$
B
$\{\boldsymbol{c}_i, \boldsymbol{e}_i\}_{i=1}^{\tau}$
S
$\{\boldsymbol{c}_i\}_{i=1}^{\tau}$
Π_S
$\{v_{\pi(i)}\}_{i=1}^{\tau}$
r
F
R
A
Key material
Computation
$k, \boldsymbol{c}_{a_j}, \mathrm{dk}_R$
$r \leftarrow \texttt{PRF}_k(V_j, \hat{r})$
Verify $\Pi_{\hat{r}}$

Fig. 6: Complete voting protocol. A voter V_j gives a vote v_j to their computer D. The value $\boldsymbol{d}_j$ is the opening of the commitment $\boldsymbol{c}_j$. The public keys for commitments and encryption are assumed known to all parties. Signatures are omitted: D signs the vote to be verified by the ballot box B and the return code server R, while R signs the incoming votes and sends the signature in return, via B, to D to confirm that the vote is received. Both B and R sends the commitments of the votes to authorities A to verify consistent views. After all votes are cast, B forwards them to the shuffle server S, stripping away the voters id's and signatures.

these values, together with its identity and every public key and commitment used to create the proofs. We note that the voter's identity and relevant keys and commitments are included in the proofs used. (This is not an artifact of making the security proof work, but it prevents real-world attacks.)

The computer sends the commitments, encryptions, proofs and signature to the ballot box. The ballot box verifies the signature and the proofs. Then it sends everything to the return code generator.

The return code generator verifies the signature and the proofs. Then it decrypts the opening of $\boldsymbol{e}_{\hat{r}}$ and computes the return code from $\hat{r}$. It hashes everything and creates a signature on the hash. It sends the return code to the voter's phone and the signature to the ballot box. The ballot box verifies the return code generator's signature on the hash. Then it sends the return code generator's signature to the voter's computer.

The voter's computer verifies the return code generator's signature and then shows the hash and the signature to the voter as the transcript. The voter checks that the computer accepts the ballot as cast. When the phone displays the return code r, the voter accepts the ballot as cast if (v_j, r) is in the return code table.

Tallying. When the tally phase begins, the ballot box sends everything from every successful ballot casting to the auditors. It extracts the commitments to the ballots and the encrypted openings (without any proofs), organizes them into a sorted list of commitment-ciphertext pairs and sends the sorted list to the shuffle server. The return code generator sends everything from every successful ballot casting to the auditors. The shuffle server receives a sorted list of commitment-ciphertext pairs from the ballot box. It hashes the list and sends a hash to the auditors. The shuffle server waits for the auditors to accept provisionally.

An auditor receives data from the ballot box and the return code generator, and a hash from the shuffle server. If the data from the ballot box and the return code generator agree, the auditor extracts the sorted commitment-ciphertext pairs from the data, hashes it and compares the result with the hash from the shuffle server. If it matches, the auditor provisionally accepts.

When the shuffle server receives the provisional accept, it decrypts the commitment openings and verifies that the openings are valid. For any invalid opening, it sends the commitment-ciphertext pair and the opening to the auditors. It then sorts the ballots and creates a shuffle proof for the ballots and the commitments. It then counts the ballots to get the election result and sends the ballots, the shuffle proof and the result to the auditors.

An auditor receives the ballots, the shuffle proof and the result from the shuffle server. It verifies the proof and that the election result is correct. It extracts the hashes (but not the signatures) signed by the return code generator from the ballot box data and creates a sorted list of hashes. It signs the hash list and the result and send both signature and hash list to the shuffle server. Once the shuffle server has received signatures and hash lists from every auditor, it verifies that the hash lists are identical and that the signatures verify. It then outputs the result, the hash list and the auditors' signatures as the transcript.

Verification. The voter has the transcript output by the voter's computer and the transcript output by the shuffle server. It first verifies that the hash from the computer's transcript is present in the shuffle server's hash list. Then it verifies all the signatures. If everything checks out, the voter accepts.

6 Performance

As outlined in our construction, we are nesting the commitment scheme of [BDL$^+$18] into the encryption scheme of [LN17]. To determine secure while not enormously big parameters for our scheme, we need to first make sure that we have sufficiently large parameters to ensure both binding and hiding of the commitments for which we will use the "optimal" parameter set of [BDL$^+$18] (but with twice the standard deviation to keep the probability of abort in the rejection sampling down to 3 trials for the proofs of linearity) which is both

Parameter	Explanation	Constraints
N, δ	Degree of polynomial $X^N + 1$ in R	$N \geq \delta \geq 1$, where N, δ powers of two
p	Modulus for commitments	Prime $p = 2\delta + 1 \mod 4\delta$
β_∞	∞-norm bound of certain elements	Choose β_∞ such that $\beta_\infty < p^{1/\delta}/\sqrt{\delta}$
σ_{C}	Standard deviation of discrete Gaussians	Chosen to be $\sigma_{\mathrm{C}} = 22 \cdot \nu \cdot \beta_\infty \cdot \sqrt{kN}$
k	Width (over R_p) of commitment matrix	
n	Height (over R_p) of commitment matrix	
ν	Maximum l_1-norm of elements in $\mathcal{C}$	
$\mathcal{C}$	Challenge space	$\mathcal{C} = \left\{c \in R_p \mid \|c\|_\infty = 1, \|c\|_1 = \nu\right\}$
$\bar{\mathcal{C}}$	The set of differences $\mathcal{C} - \mathcal{C}$ excluding 0	$\bar{\mathcal{C}} = \{c - c' \mid c \neq c' \in \mathcal{C}\}$
D_{β_∞}	Set of elements of ∞-norm at most β_∞	$D_{\beta_\infty} = \{x \in R_p \mid \|x\|_\infty \leq \beta_\infty\}$
σ_{E}	Standard deviation of discrete Gaussians	Chosen to be $\sigma_{\mathrm{E}} = 11 \cdot \nu \cdot \sqrt{\kappa N(3 + \beta_\infty)}$
κ	Dimension of message space in encryption	Equal to the length of randomness k
ℓ	Dimension the encryption matrix	Equal to the size of the commitments $k - n$
λ	Dimension of public $\boldsymbol{u}$ in $\boldsymbol{T\mu} = \boldsymbol{u}$	Equal to the height $n + 1$ of the commitment matrix
q	Modulus for encryption	Must choose prime q such that $q > 24\sigma_{\mathrm{E}}^2$ and $q > 2p(2\ell \cdot N^2 \cdot {\beta_\infty}^2 + N + 1)$ and $q = 2\delta + 1 \mod 4\delta$
τ	Total number of votes	For soundness we need $(\tau^\delta + 1)/\lvert R_p\rvert < 2^{-128}$

Table 1: Parameters for the commitment and verifiable encryption schemes.

computationally binding and hiding (see Table 2). The LWE-estimator [APS15] estimates at least 100 bits of security with these parameters. We then instantiate the verifiable encryption scheme with compatible parameters, which is possible due to our generalization of [LN17]. The verifiable encryption scheme will then yield decryptions with an ∞-norm that is way below the bound for which the commitment scheme is binding, so any valid decryption which differs from the original vote would break the binding of the commitment scheme. In general, the instantiation of the encryption scheme offers much higher security than the commitment scheme, but the choice of parameters are restricted by the constraints from combining it with the commitments.

6.1 Size

Size of the Votes. Note that each ciphertext $\boldsymbol{e}$ includes both the encrypted opening $(\boldsymbol{v}, \boldsymbol{w})$ and the proof of valid opening $(c, \boldsymbol{z})$. Using a lattice based signature scheme like Falcon-768 [PFH+17], we have signatures of size ≈ 1 KB. The voter verifiability protocol requires a commitment, an encryption + proof, and a proof of linearity. It follows that a vote $(\boldsymbol{c}_j, \boldsymbol{e}_j, \boldsymbol{c}_{\hat{r}}, \boldsymbol{e}_{\hat{r}}, \Pi_{\hat{r}})$ is of total size ≈ 240 KB, which means that, for τ voters, the ballot box $\mathcal{B}$ receives 240τ KB of data.

Size of the Shuffle Proof. Our shuffle protocol is a $4 + 3\tau$-move protocol with elements from R_p. Each element in R_p has at most N coefficients of size at most p, and hence, each R_p-element has size at most $N \log p$ bits. For every R_p-vector that follows a Gaussian distribution with standard deviation σ we assume that we can represent the element using $N \cdot \log(6\sigma)$ bits. Every element from $\mathcal{C}$ will be assumed to be representable using at most $2N$ bits.

Parameter	Commitment (I)	Encryption (III)
N	1024	1024
p	$\approx 2^{32}$	$\approx 2^{32}$
q	-	$\approx 2^{56}$
β_∞	1	1
σ	$\sigma_{\mathrm{C}} \approx 54000$	$\sigma_{\mathrm{E}} \approx 54000$
ν	36	36
δ	2	2
k	3	-
n	1	-
ℓ	-	2
κ	-	3
λ	-	2
Proof	9.4 KB	42.4 KB
Primitive	8.2 KB	64.5 KB

Table 2: Parameters for the commitments by Baum et al. [BDL+18] and verifiable encryption scheme by Lyubashevsky and Neven [LN17].

We analyze how much data we have to include in each step of the shuffling protocol in Figure 2. Using the Fiat-Shamir transform [FS87], we can ignore the challenge-messages from the verifier. The prover ends up sending 1 commitment, 1 ring-element and 1 proof of linearity per vote. Using the parameters from Table 2, we get that the shuffle proof is of total size $\approx 22\tau$ KB.

6.2 Timings

We collected performance figures from our prototype implementation written in C to estimate the runtime of our scheme. Estimates are based on Table 2 and the implementation was benchmarked on an Intel Skylake Core i7-6700K CPU running at 4GHz without TurboBoost using `clang` 12.0 and FLINT 2.7.1 [HJP13]. Timings are available in Table 3, and the source code can be found on GitHub [12].

Our Scheme:	Commit	Open	Encrypt	Verify	Decrypt	Shuffle
Time	1.1 ms	1.2 ms	208 ms	39 ms	6 ms	27τ ms

Table 3: Timings for cryptographic operations. Numbers were obtained by computing the average of 10^4 consecutive executions of an operation measured using the cycle counter available in the platform.

[12] github.com/dfaranha/lattice-voting-ctrsa21

Elementary Operations. Multiplication in R_p and R_q is usually implemented when $p \equiv q \equiv 1 \bmod (2N)$ and $X^N + 1$ splits in N linear factors, for which the Number-Theoretic Transform is available. Unfortunately, Lemma 1 restricts parameters and we instead adopt $p \equiv q \equiv 5 \bmod 8$ [LN17]. In this case, $X^N + 1$ splits in two $N/2$-degree irreducible polynomials $(X^{N/2} \pm r)$ for r a modular square root of -1. This gives an efficient representation for $a = a_1 X^{N/2} + a_0$ using the Chinese Remainder Theorem: $CRT(a) = (a \pmod{X^{N/2} - r}, a \pmod{X^{N/2} + r})$. Even though the conversions are efficient due to the choice of polynomials, we sample ring elements directly in this representation whenever possible. As in [LS18], we implement the base case for degree $N/2$ using FLINT for polynomial arithmetic [HJP13]. We use SHA256 for hashing to generate challenges.

Commitment. A commitment is generated by multiplying the matrix $\boldsymbol{B}$ by a vector $\boldsymbol{r}_m$ over R_p and finally adding the message m to the second component in the CRT domain. Computing and opening a commitment takes 0.9 ms and 1.2 ms, respectively, and sampling randomness $\boldsymbol{r}_m$ takes only 0.2 ms.

Verifiable Encryption. Verifiable encryption needs to sample vectors according to a discrete Gaussian distribution. For an R_q element with standard deviation $\sigma_{\mathrm{E}} \approx 2^{15.7}$ (for the encryption scheme), the implementation from COSAC [ZSS20] made available for $\sigma = 2^{17}$ samples 1024 integers in 0.12 ms using very small precomputation tables. Each encryption iteration takes 69 ms and, because we expect to need 3 attempts to generate one valid encryption (line 8 in Figure 3), the total time of encryption is around 208 ms. For verification, 39 ms are necessary to execute a test; and 6 ms are required for the actual decryption.

Shuffle Proof. The shuffle proof operates over R_p and is thus more efficient. Sampling uses the same approach as above for σ_{C} from the commitment scheme. Benchmarking includes all samplings required inside the protocol, the commitment, the proof of linearity and, because we expect to need 3 attempts to generate each of the proofs of linearity to the cost of 7.5 ms, amounts to 27τ ms for the entire proof, omitting the communication cost.

6.3 Comparison

We briefly compare our scheme with the scheme by del Pino et al. [dLNS17] from CCS 2017 in Table 4. We note that the scheme in [dLNS17] requires at least $\xi \geq 2$ authorities to ensure ballot privacy, where at least one authority must be honest. The authorities run the proof protocol in parallel, and the time they need to process each vote is ≈ 5 times slower per vote than in our scheme. We only need one party to compute the shuffle proof, where we first decrypt all votes and then shuffle. Our proof size is at least 14 KB smaller per vote when $\xi = 2$, that is, a saving of 40 %, and otherwise much smaller in comparison for $\xi \geq 3$. We further note that both implementations partially rely on FLINT

Comparison	Vote Size	Voter Time	Proof Size	Prover Time
Our Scheme:	120 KB	209 ms	22τ KB	33τ ms
CCS 2017 [dLNS17]:	20ξ KB	9 ms	$18\xi\tau$ KB	150τ ms

Table 4: Comparing our scheme with the yes/no voting scheme in [dLNS17]

for polynomial arithmetic and were benchmarked on Intel Skylake processors. A significant speedup persists after correcting for clock frequency differences.

For a fair comparison, we only included the size and timings of the commitment of the vote and the encrypted openings from our scheme. In practice, the size and timings of the voter will be twice of what it is in the table, because of the return code mechanism, which is not a part of [dLNS17]. This has no impact on the decryption and shuffle done by the prover. The work done by the voter is still practical. For [dLNS17] to be used in a real world election, they would need to include an additional mechanism for providing voter verifiability, similar to the one we have constructed.

Finally, we note that [dLNS17] can be extended from yes/no voting to votes consisting of strings of bits. However, the size and timings of such an extension will be linear in the length of the bit-strings, and our scheme would do even better in comparison, as we can handle votes encoded as arbitrary ring-elements.

Thanks

We thank Andreas Hülsing and the anonymous reviewers for helpful comments.

References

Adi08. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008*, pages 335–348. USENIX Association, July / August 2008.

ALS20. Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler. Practical product proofs for lattice commitments. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 470–499. Springer, Heidelberg, August 2020.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

BBC$^+$18. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 669–699. Springer, Heidelberg, August 2018.

BCG$^+$15. David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society Press, May 2015.

BDL$^+$18. Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 368–385. Springer, Heidelberg, September 2018.

BG14. Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 28–47. Springer, Heidelberg, February 2014.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

BHM20. Xavier Boyen, Thomas Haines, and Johannes Müller. A verifiable and practical lattice-based decryption mix net with external auditing. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 336–356. Springer, Heidelberg, September 2020.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic LWE based E-voting scheme. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 245–265. Springer, Heidelberg, 2016.

Cha81. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

CMM19. Núria Costa, Ramiro Martínez, and Paz Morillo. Lattice-based proof of a shuffle. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *FC 2019 Workshops*, volume 11599 of *LNCS*, pages 330–346. Springer, Heidelberg, February 2019.

dLNS17. Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. Practical quantum-safe voting from lattices. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1565–1581. ACM Press, October / November 2017.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

Gjø11. Kristian Gjøsteen. The norwegian internet voting protocol. In *E-Voting and Identity - Third International Conference, VoteID 2011*, pages 1–18, 2011.

GL16. Kristian Gjøsteen and Anders Smedstuen Lund. An experiment on the security of the Norwegian electronic voting protocol. *Annals of Telecommunications*, pages 1–9, 2016.

GLP12. Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, Heidelberg, September 2012.

GS17. Kristian Gjøsteen and Martin Strand. A roadmap to fully homomorphic elections: Stronger security, better verifiability. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *FC 2017 Workshops*, volume 10323 of *LNCS*, pages 404–418. Springer, Heidelberg, April 2017.

HJP13. W. Hart, F. Johansson, and S. Pancratz. FLINT: Fast Library for Number Theory, 2013. Version 2.4.0, `http://flintlib.org`.

HR16. Feng Hao and Peter Y. A. Ryan, editors. *Real-World Electronic Voting: Design, Analysis and Deployment.* CRC Press, 2016.

LN17. Vadim Lyubashevsky and Gregory Neven. One-shot verifiable encryption from lattices. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 293–323. Springer, Heidelberg, April / May 2017.

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

LPT19. Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. Trapdoor commitments in the SwissPost e-voting shuffle proof, 2019.

LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, June 2015.

LS18. Vadim Lyubashevsky and Gregor Seiler. Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 204–224. Springer, Heidelberg, April / May 2018.

Lyu09. Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Heidelberg, December 2009.

Lyu12. Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Heidelberg, April 2012.

Nef01. C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 116–125. ACM Press, November 2001.

PFH$^+$17. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

Scy. Scytl. Scytl sVote, complete verifiability security proof report - software version 2.1 - document 1.0. `https://www.post.ch/-/media/post/evoting/dokumente/complete-verifiability-security-proof-report.pdf`.

Str19. Martin Strand. A verifiable shuffle for the GSW cryptosystem. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 165–180. Springer, Heidelberg, March 2019.

ZSS20. Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. COSAC: COmpact and scalable arbitrary-centered discrete gaussian sampling over integers. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 284–303. Springer, Heidelberg, 2020.

Appendix

A Security Proof of the Shuffle Protocol

We now show how to prove correctness and the zero-knowledge property of the protocol Π_{Shuffle} as described in Figure 2.

Theorem 2. *The shuffle protocol Π_{Shuffle} as described in Figure 2 is complete for the relation $\mathcal{R}_{\text{Shuffle}}$ if $\tau \ll |R_p|$ and Π_{Lin} is complete.*

Proof. Due to the size of R_p there must always exist such a value ρ. Furthermore, the instances of Π_{Lin} always succeed due to the completeness of the respective proof if the s_i indeed fulfill the required linear relations. As Lemma 4 shows that the respective values for s_i always exist, the claim follows. □

We prove Special Honest Verifier Zero-Knowledge (HVZK) by a series of games, as shown in Figure 7. In the first game, we swap the HVZK protocol Π_{Lin} from Figure 1 with its simulation. Then we change the way we calculate the initial commitments $\llbracket D_i \rrbracket$ by instead sampling s_i uniformly random and computing what D_i should be. In this step we make no use of the values θ_i. Next, we let the $\llbracket D_i \rrbracket$'s be commitments $\llbracket 0 \rrbracket$, and sample random s_i's. Thus, we do not use any permutation in the last game; the Simulator is indistinguishable from the Real game. We proceed with the proofs that an adversary cannot distinguish between any of these games.

Real	Game 1	Game 2	Simulator
1 : $\theta_i \xleftarrow{\$} R_p$	1 : $\theta_i \xleftarrow{\$} R_p$	1 : $\boxed{s_i \xleftarrow{\$} R_p}$	1 : $\boxed{\llbracket D_i \rrbracket \leftarrow \llbracket 0 \rrbracket}$
2 : $D_i \leftarrow (1)$	2 : $D_i \leftarrow (1)$	2 : $\theta_i \xleftarrow{\$} R_p$	2 : $s_i \xleftarrow{\$} R_p$
3 : Send $\llbracket D_i \rrbracket$	3 : Send $\llbracket D_i \rrbracket$	3 : $\boxed{D_i \leftarrow (8)}$	3 : Send $\llbracket D_i \rrbracket$
4 : $s_i \leftarrow (2)$	4 : $s_i \leftarrow (2)$	4 : Send $\llbracket D_i \rrbracket$	4 : Send s_i
5 : Send s_i	5 : Send s_i	5 : Send s_i	5 : Sim
6 : SHVZK	6 : $\boxed{\textbf{Sim}}$	6 : Sim	

Fig. 7: Honest Verifier Zero-Knowledge games. We denote by $x \leftarrow (i)$ that x was computed according to equation (i). The steps that changes from game to game are indicated with boxes.

Lemma 6. *If there exists an adversary that can distinguish between Real and Game 1 in Figure 7, then there exists an adversary that can break the HVZK property of Π_{Lin}.*

Proof. This follows because the only difference between Real and Game 1 is that we simulate the last step, which is exactly the zero-knowledge proof. □

Lemma 7. *The distribution of the transcripts* $(\{[\![D_i]\!]\}, \beta, \{s_i\})$ *produced by* Game 1 *and* Game 2 *are perfectly indistinguishable.*

Proof. We give the argument for one of the equations. The argument is identical for the remaining equations. In Game 1 we sample $\theta_1 \xleftarrow{\$} R_p$, compute

$$[\![D_1]\!] = [\![\theta_1 \hat{M}_1]\!],$$

and then compute s_1 according to (3). Since θ_1 is uniform, this is a commitment to a uniformly random value. Also, we know that s_1 is uniformly distributed.

In Game 2 we sample a uniformly random $s_1 \xleftarrow{\$} R_p$ and then compute

$$[\![D_1]\!] = [\![\beta \hat{M}_1 + s_1 M_1]\!]. \tag{8}$$

Now, s_1 is uniformly random, so $[\![D_1]\!]$ is a commitment to a uniformly random value. Hence, it follows that the transcripts are perfectly indistinguishable. □

Lemma 8. *If there exists an adversary $\mathcal{A}$ that can distinguish between the transcripts of* Game 2 *and* Simulator*, then there exists an adversary $\mathcal{A}'$ who breaks the hiding property of the commitment scheme.*

Proof. The proof is shown in Figure 8.

We use a standard distinguishing game. A commitment oracle will then commit to random values $m \in R_p$ or to 0. The oracle will send the challenges $\{\boldsymbol{c}_{b,i}\}$ to $\mathcal{A}'$ where b is a bit indicating commitments to 0 or m if b is 0 or 1, respectively. Then $\mathcal{A}'$ will pick random values s_i and send the transcript $(\{\boldsymbol{c}_{b,i}\}, \beta, \{s_i\})$ to the distinguishing adversary $\mathcal{A}$. We then use the distinguishing power of $\mathcal{A}$ to decide on a bit b' and send this back to the commitment challenger.

We now show that the $\boldsymbol{c}_{b,i}$'s are distributed as $\mathcal{A}$ expected. When querying $\mathcal{A}$ on the transcripts, we use the challenges $\boldsymbol{c}_{b,i} \leftarrow \mathcal{C}(m_i, r_i)$ for a uniformly random m_i, but $\mathcal{A}$ expects the commitments $[\![D_i]\!]$ from (8). For the first equation of (8), we note that the committed message is $\beta \hat{M}_1 + s_1 M_1$. Since each s_i is a uniform sample from R_p, this expression is a uniformly random sample from R_p. Hence, the distributions of $\boldsymbol{c}_{1,i}$ and $[\![D_i]\!]$ (from (8)) are identical. □

This means that we can simulate a transcript of the real protocol without knowing any of the private information known to $\mathcal{P}$. We summarize this result in a theorem.

Theorem 3. *Assuming that Π_{Lin} is an HVZK proof and that the used commitment scheme is hiding, then the transcripts of* Real *and* Simulator *in Figure 7 are indistinguishable.*

Proof. This is true by combining Lemma 6, 7 and 8. □

Adversary Breaking the Hiding Property

```
Commitment Oracle
for $i = 1, \dots \tau$ do
    $m \xleftarrow{\$} R_p$
    $\boldsymbol{r} \xleftarrow{\$} D^k_{\beta_\infty}$
    $\boldsymbol{c}_{0,i} \leftarrow \mathcal{C}(0, \boldsymbol{r})$
    $\boldsymbol{c}_{1,i} \leftarrow \mathcal{C}(m, \boldsymbol{r})$
$b \leftarrow \{0, 1\}$
$\xrightarrow{\{\boldsymbol{c}_{b,i}\}_{i=1}^{\tau}}$
    Using Adversary
    $\mathcal{A}'$                                   $\mathcal{A}$
    $s_i \xleftarrow{\$} R_p$
    $\beta \xleftarrow{\$} R_p$
                  $\xrightarrow{\{\boldsymbol{c}_{b,i}\}, \beta, \{s_i\}}$
                                          $b' \leftarrow \mathcal{A}(\{\boldsymbol{c}_{b,i}\}, \beta, \{s_i\})$
                  $\xleftarrow{b'}$
$\xleftarrow{b'}$
if $b' = b$ then
    Return 1
Return 0
```

Fig. 8: Using an adversary $\mathcal{A}$ who can distinguish between Game 2 and Simulator in Figure 7, we can construct an adversary $\mathcal{A}'$ who can break the hiding property of the commitment scheme.

B Security of the Voting Protocol

Our return code mechanism does not fit the standard voting scheme syntax security definitions such as [BCG+15]. The usual solution is to define *ad hoc* security definitions (see e.g. [Scy] and [Gjø11]). Note that protocols with this [Gjø11] and similar architectures have been deployed in large-scale national elections.

We follow the standard approach for voting system analysis, which is to consider a voting system to be fairly simple cryptographic protocol built on top of a *cryptographic voting scheme*, which is in some sense similar to a public key cryptosystem with some very specific functionality.

We begin by describing a *verifiable* cryptographic voting scheme *with return codes*, explain how our voting system can be described as a simple protocol on top of such a cryptographic voting scheme. We define security notions for this cryptosystem, sketch the security proof, and then informally discuss the voting protocol and its security properties in terms of the cryptosystem's security.

B.1 Verifiable Voting Schemes with Return Codes

A *verifiable cryptographic voting scheme* in our architecture is usually defined in terms of algorithms for the four tasks election setup, casting ballots, counting cast ballots and verifying the count. To support return codes, we need algorithms for two more tasks: voter registration and pre-code computation.

The *setup* algorithm `Setup` outputs a *public key* `pk`, a *decryption key* `dk` and a *code key* `ck`.

The *register* algorithm `Reg` takes a public key `pk` as input and outputs a *voter verification key* `vvk`, a *voter casting key* `vck` and a function f from ballots to pre-codes.

The *cast* algorithm `Cast` takes a public key `pk`, a voter casting key `vck` and a *ballot* v, and outputs an *encrypted ballot* ev and a *ballot proof* Π^v.

The *code* algorithm `Code` takes a code key `ck`, an encrypted ballot ev and a proof Π^v as input and outputs a pre-code $\hat{r}$ or $\perp$.

The *count* algorithm `Count` takes a decryption key `dk` and a sequence of encrypted ballots $ev_1, ev_2, \dots, ev_{l_t}$, and outputs a sequence of ballots $v_1, v_2, \dots, v_{l_t}$ and a proof Π^c, or $\perp$.

The *verify* algorithm `Verify` takes a public key `pk`, a sequence of encrypted ballots $ev_1, ev_2, \dots, ev_{l_t}$, a sequence of ballots $v_1, v_2, \dots, v_{l_t}$ and a proof Π^c, and outputs 0 or 1.

A cryptographic voting scheme is *correct* if for any $(\mathtt{pk}, \mathtt{dk}, \mathtt{ck})$ output by `Setup` and any $(\mathtt{vvk}_1, \mathtt{vck}_1, f_1), \dots, (\mathtt{vvk}_{l_V}, \mathtt{vck}_{l_V}, f_{l_V})$ output by $\mathtt{Reg}(\mathtt{pk})$, any ballots $v_1, \dots, v_{l_V}$, any (ev_i, Π_i^v) output by $\mathtt{Cast}(\mathtt{pk}, \mathtt{vck}_i, v_i)$, $i = 1, 2, \dots, l_V$, and any $(v'_1, \dots, v'_{l_V}, \Pi^c)$ output by $\mathtt{Count}(\mathtt{dk}, ev_1, \dots, ev_{l_V})$, then:

- $\mathtt{Code}(\mathtt{ck}, \mathtt{vvk}_i, ev_i, \Pi_i^v) = f_i(v_i)$,
- $\mathtt{Verify}(\mathtt{pk}, ev_1, \dots, ev_{l_V}, v'_1, \dots, v'_{l_V}, \Pi^c) = 1$, and
- $v_1, \dots, v_{l_V}$ equals $v'_1, dots, v'_{l_V}$, up to order.

We further require that the distribution of ev_i only depends on $\texttt{pk}$ and v_i, not $\texttt{vck}_i$. Also, whether $\texttt{Count}$ outputs $\perp$ does not depend on the order of the encrypted ballots, and if $\texttt{Count}$ outputs $\perp$ for some list $(ev_1, \ldots, ev_{l_V})$ of encrypted ballots, $\texttt{Count}$ outputs $\perp$ for any other list with $(ev_1, \ldots, ev_{l_V})$ as a prefix.

B.2 Our Scheme

We summarize the scheme from Section 5 in the above terms. The commitment algorithms are $(\texttt{KeyGen}_C, \texttt{Com}(,)\texttt{Open}())$, the verifiable encryption algorithms are $(\texttt{KeyGen}_{VE}, \texttt{Enc}_{VE}, \texttt{Ver}_{VE}, \texttt{Dec}_{VE})$.

- The *setup* algorithm computes $\texttt{pk}_C \leftarrow \texttt{KeyGen}_C$, $(\texttt{pk}_V, \texttt{dk}_V) \leftarrow \texttt{KeyGen}_{VE}$, $(\texttt{pk}_R, \texttt{dk}_R) \leftarrow \texttt{KeyGen}_{VE}$. The public key $\texttt{pk} = (\texttt{pk}_C, \texttt{pk}_V, \texttt{pk}_R)$, the decryption key is $\texttt{dk} = (\texttt{pk}_C, \texttt{dk}_V)$ and the code key is $\texttt{ck} = (\texttt{pk}_C, \texttt{pk}_V, \texttt{dk}_R)$.
- The *register* algorithm takes $\texttt{pk} = (\texttt{pk}_C, \texttt{pk}_V, \texttt{pk}_R)$ as input. It samples $a \xleftarrow{\$} R_p$ and computes $(\boldsymbol{c}_a, d_a) \leftarrow \texttt{Com}(\texttt{pk}_C, a)$. The voter verification key is $\texttt{vvk} = \boldsymbol{c}_a$, the voter casting key is $(a, \boldsymbol{c}_a, d_a)$, and the function f is $v \mapsto v + a$.
- The *cast* algorithm takes $\texttt{pk} = (\texttt{pk}_C, \texttt{pk}_V, \texttt{pk}_R)$, $\texttt{vck} = (a, \boldsymbol{c}_a, d_a)$ and v as input. It computes $(\boldsymbol{c}, d) \leftarrow \texttt{Com}(\texttt{pk}_C, v)$, $\hat{r} \leftarrow a + v$, $(\boldsymbol{c}_{\hat{r}}, d_{\hat{r}}) \leftarrow \texttt{Com}(\texttt{pk}_C, \hat{r})$, $\Pi_{\hat{r}}^{lin}$is a proof that $\boldsymbol{c} + \boldsymbol{c}_a$ (which is a commitment to $v + a$) and $\boldsymbol{c}_{\hat{r}}$ satisfy the relation $v + a = \hat{r}$. Then it encrypts $\boldsymbol{e} = (\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}) \leftarrow \texttt{Enc}_{VE}(\texttt{pk}_V, d)$ and $\boldsymbol{e}_{\hat{r}} = (\boldsymbol{v}_{\hat{r}}, \boldsymbol{w}_{\hat{r}}, c_{\hat{r}}, \boldsymbol{z}_{\hat{r}}) \leftarrow \texttt{Enc}_{VE}(\texttt{pk}_R, d_{\hat{r}})$. The encrypted ballot is $ev = (\boldsymbol{c}, d, \boldsymbol{v}, \boldsymbol{w}, c)$, while the ballot proof is $\Pi^v = (\boldsymbol{z}, \boldsymbol{c}_{\hat{r}}, \boldsymbol{e}_{\hat{r}}, \Pi_{\hat{r}}^{lin})$.
- The *code* algorithm takes $\texttt{ck} = (\texttt{pk}_C, \texttt{pk}_V, \texttt{dk}_R)$, a voter verification key $\texttt{vvk}$, an encrypted ballot $ev = (\boldsymbol{c}, d, \boldsymbol{v}, \boldsymbol{w}, c)$ and a ballot proof $\Pi^v = (\boldsymbol{z}, \boldsymbol{c}_{\hat{r}}, \boldsymbol{e}_{\hat{r}}, \Pi_{\hat{r}}^{lin})$ as input. It verifies $\Pi_{\hat{r}}^{lin}$, and then verifies $(\boldsymbol{v}, \boldsymbol{w}, c, \boldsymbol{z}))$ and $\boldsymbol{e}_{\hat{r}}$ using $\texttt{Ver}_{VE}$. If any verification fails, it outputs $\perp$. It then decrypts $d_{\hat{r}} \leftarrow \boldsymbol{e}_{\hat{r}}$ and recovers $\hat{r}$ from $\boldsymbol{c}_{\hat{r}}$ and $d_{\hat{r}}$, and outputs $\hat{r}$.
- The *count* algorithm takes as input $\texttt{dk} = (\texttt{pk}_C, \texttt{dk}_V)$ and encrypted ballots $(\boldsymbol{c}_1, \boldsymbol{v}_1, \boldsymbol{w}_1, c_1), \ldots, (\boldsymbol{c}_{l_t}, \boldsymbol{v}_{l_t}, \boldsymbol{w}_{l_t}, c_{l_t})$. It computes $d_i \leftarrow \texttt{Dec}_{VE}(\texttt{dk}_V, \boldsymbol{v}_i, \boldsymbol{w}_i, c_i)$ and recovers v_i' from $\boldsymbol{c}_i$ and d_i. If any decryption fails, it outputs $\perp$. Otherwise, it chooses a random permutation π on $\{1, 2, \ldots, l_t\}$, sets $v_{\pi(i)} = v_i'$, and creates a proof of shuffle of known values Π^c. It outputs $v_1, v_2, \ldots, v_{l_t}$ and Π^c.
- The *verify* algorithm takes as input $\texttt{pk} = (\texttt{pk}_C, \texttt{pk}_V, \texttt{pk}_R)$, encrypted ballots $(\boldsymbol{c}_1, \boldsymbol{v}_1, \boldsymbol{w}_1, c_1), \ldots, (\boldsymbol{c}_{l_t}, \boldsymbol{v}_{l_t}, \boldsymbol{w}_{l_t}, c_{l_t})$, ballots $v_1, v_2, \ldots, v_{l_t}$ and a count proof Π^c. It verifies that Π^c is a correct proof of shuffle of known values for $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{l_t}$ and $v_1, v_2, \ldots, v_{l_t}$. It outputs 1 if verification holds, otherwise it outputs 0.

It is straight-forward to verify that the scheme is correct.

B.3 Our Voting Protocol

Recall the protocol described in Figure 6.

In the *setup phase*, a trusted set of players run the setup algorithm `Setup`. The derived public key `pk` is given to every player, the decryption key `dk` is given to the shuffler S and the code key `ck` is given to the return code generator R.

In the *registration phase*, a set of trusted players run the register algorithm `Reg` to generate per-voter keys $(\mathtt{vvk}, \mathtt{vck}, f)$ for the voter V, making every verification key public and giving `vck` to the voter's computer. Then the return code generator chooses key k for `PRF`, and a set of trusted players compute the return code table $\{(v_i, \mathtt{PRF}_k(V, f(v_i))\}$ for a relatively small set of ballots $v_1, \ldots, v_\omega$. The voter gets the return code table.

In the *casting phase*, the voter V instructs the voter's computer D which ballot to cast. The voter's computer D runs the casting algorithm `Cast` and sends the encrypted ballot and the ballot proof to the ballot box B. (In practice, the ballot box will verify the ballot proofs, but it is not necessary.) The ballot box B sends the encrypted ballot and the ballot proof to the return code generator R, who runs the code algorithm `Code` to get the precode $\hat{r}$. It computes the return code $r \leftarrow \mathtt{PRF}_k(\hat{r})$ and sends r to the voter's phone F, which sends it on to the voter V. The voter V compares the return code to the code in its return code table, and accepts the ballot as cast if and only if the codes match.

In addition, the voter's computer will sign the encrypted ballot and ballot proof on behalf of the voter. The ballot box and the return code generator will verify this signature. The return code generator will countersign everything and return this signature to the voter's computer via the ballot box. The voter's computer will verify the countersignature. (This ensures that if the voter's computer accepts the ballot as cast, the ballot box and the return code generator agree that the ballot was indeed cast. If either of them is honest, this gives us a stronger form of integrity.)

In the casting phase, every player uses fixed-length encodings for messages, ensuring that every message of a given type has a fixed length that is public knowledge.

In the *counting phase*, the ballot box B and the return code generator R send the encrypted ballots and ballot proofs they have seen to the auditor A. If the data is consistent, the auditor A approves. The ballot box B then sorts the list of encrypted ballots and sends this to the shuffler S. (In the event that some voter has cast more than one ballot, only the encrypted ballot seen last is included.) The shuffler S uses the count algorithm `Count` to compute a list of ballots $v_1, \ldots, v_{l_t}$ and a shuffle proof, which is sent to the auditor A. The auditor A uses the verification algorithm `Verify` to verify the shuffle proof against the encrypted ballots received from B and R.

This concludes the description of the voting protocol in terms of a verifiable cryptographic voting scheme with return codes.

If some limited verifiability is desired, the ballot box B, return code generator R and auditor A may make commitments to the encrypted ballots received public. The voter's computer can be given the commitment and an opening to verify the correctness of the commitment to the voter's encrypted ballot, as well as its presence in the public record.

Note that more than one auditor can be used in the protocol.

B.4 Security Notions

Our notion of confidentiality is similar to the usual ballot box privacy notions [BCG+15]. An adversary that sees both the contents of the ballot box and the decrypted ballots should not be able to determine who cast which ballot. This should hold even if the adversary can see pre-codes, learn the code key and some voter casting keys, and insert maliciously generated ciphertexts into the ballot box.

Our notion of integrity is again fairly standard, adapted to return codes. An adversary should not be able to cause an incorrect pre-code or inconsistent decryption or non-unique decryption, even if the adversary knows all of the key material.

We define security notions for a verifiable cryptographic voting scheme using an experiment where an adversary $\mathcal{A}$ is allowed to reveal keys, make challenge queries, create ciphertexts and choose which ballots get counted. This experiment models both a left-or-right game for confidentiality, and a test query for integrity. The experiment works as follows:

– Sample $b, b'' \overset{\$}{\leftarrow} \{0,1\}$. Set L to be an empty list.
– $(\mathtt{pk}, \mathtt{dk}, \mathtt{ck}) \leftarrow \mathtt{Setup}$. For $i = 1, \dots, l_V$: $(\mathtt{vvk}_i, \mathtt{vck}_i, f_i) \leftarrow \mathtt{Reg}(\mathtt{pk})$. Send $(\mathtt{pk}, \mathtt{vvk}_1, \dots, \mathtt{vvk}_{l_V})$ to $\mathcal{A}$.
– On a *voter reveal query* i, send $(\mathtt{vck}_i, f_i)$ to $\mathcal{A}$. On a *decrypt reveal query*, send $\mathtt{dk}$ to $\mathcal{A}$. On a *code reveal query*, send $\mathtt{ck}$ to $\mathcal{A}$.
– On a *challenge query* (i, v_0, v_1), compute $(ev, \Pi^v) \leftarrow \mathtt{Cast}(\mathtt{pk}, \mathtt{vck}_i, v_b)$, $\hat{r} \leftarrow \mathtt{Code}(\mathtt{ck}, \mathtt{vvk}_i, ev, \Pi^v)$, append (i, v_0, v_1, ev, Π^v) to L and send (ev, Π^v) to $\mathcal{A}$.
– On a *chosen ciphertext query* (i, ev, Π^v), compute $\hat{r} \leftarrow \mathtt{Code}(\mathtt{ck}, \mathtt{vvk}_i, ev, \Pi^v)$. If $\hat{r} \neq \perp$, append $(i, \perp, \perp, ev, \Pi^v)$ to L. Send $\hat{r}$ to $\mathcal{A}$.
– On a *count query* $(j_1, \dots, j_{l_s})$, with

$$L = ((i_1, v_{0,1}, v_{1,1}, ev_1, \Pi_1^v), \dots, (i_{l_t}, v_{0,l_t}, v_{1,l_t}, ev_{l_t}, \Pi_{l_t}^v)),$$

compute *result* $\leftarrow \mathtt{Count}(\mathtt{dk}, ev_{j_1}, \dots, ev_{j_{l_s}})$ and send *result* to $\mathcal{A}$.
– On a *test query* $(j_1, \dots, j_{l_s}, v_1, \dots, v_{l_s}, \Pi^c)$, compute *result* $\leftarrow$ $\mathtt{Verify}(\mathtt{pk}, ev_{j_1}, \dots, ev_{j_{l_s}}, v_1, \dots, v_{l_s}, \Pi^c)$ and send *result* to $\mathcal{A}$.

Eventually, the adversary outputs a bit b'.

Confidentiality fails trivially for the usual reasons, and in particular, the count trivially reveals the challenge bit unless the left hand ballots and the right-hand ballots are identical, up to order. (Recall that the adversary should figure out who cast which ballots, not what ballots were cast.) For executions where confidentiality fails trivially, we should not count the adversary's answer towards the advantage, so we will compare the adversary's guess with a secret random bit. Integrity can fail, either if pre-codes are incorrect, if an outcome verifies as correct but is inconsistent with the challenge ciphertexts, or if there is no unique decryption.

We define events related to confidentiality and integrity. Let E_g be the event that $b = b'$ and let E_r be the event that $b'' = b'$. Let E_f denote the event that an execution is *fresh*, which is true if the following are satisfied: there is no decrypt reveal query; for any i, there is either no challenge query, or at most one challenge query and no voter reveal query or chosen ciphertext query; if there is a count query where *result* $\neq \bot$, then the sequence $(v_{0,j_1}, \ldots, v_{0,j_{l_s}})$ equals $(v_{1,j_1}, \ldots, v_{1,j_{l_s}})$, up to order.

Let F_i (incorrect pre-code) be the event that for some chosen ciphertext query (i, ev, Π^v) where $\texttt{Code}(\texttt{ck}, \texttt{vvk}_i, ev, \Pi^v) = \hat{r} \neq \bot$, we have that either $\texttt{Count}(\texttt{dk}, ev) = \bot$ or $\texttt{Count}(\texttt{dk}, ev) = (v, \Pi^c)$ and $f_i(v) \neq \hat{r}$. Let F_c (count failure) be the event that a count query gets *result* $= \bot$. Let F_d (inconsistent decryption) be the event that a test query $(j_1, \ldots, j_{l_s}, v_1, \ldots, v_{l_s}, \Pi^c)$ with $L = ((i_1, v_{0,1}, v_{1,1}, ev_1, \Pi_1^v), \ldots, (i_{l_t}, v_{0,l_t}, v_{1,l_t}, ev_{l_t}, \Pi_{l_t}^v))$ gets *result* $= 1$ and there is no permutation π on $\{1, 2, \ldots, l_s\}$ such that $v_{b,k} = \bot$ or $v_{b,k} = v_{\pi(k)}$ for $k = 1, 2, \ldots, l_s$. Let F_u (no unique decryption) be the event that two test queries $(j_1, \ldots, j_{l_s}, v_1, \ldots, v_{l_s}, \Pi^c)$ and $(j_1, \ldots, j_{l_s}, v'_1, \ldots, v'_{l_s}, \Pi^{c\prime})$ both get *result* $= 1$, but there is no permutation π on $\{1, 2, \ldots, l_s\}$ such that $v_k = v'_{\pi(k)}$ for $k = 1, 2, \ldots, l_s$.

The advantage of the adversary is

$$\max\{2|\Pr[E_f \land E_g] + \Pr[\neg E_f \land E_r] - 1/2|, \Pr[F_i \lor F_c \lor F_d \lor F_u]\}.$$

B.5 Security Proof Sketch

We briefly sketch how a proof to bound the advantage of an adversary against the cryptographic voting scheme in terms of adversaries against the shuffle of known values, the underlying commitment scheme or the related linearity proofs, or the verifiable encryption scheme.

Confidentiality events We begin by analyzing the confidentiality events. Note that $\Pr[\neg E_f \land E_r] = (1 - \Pr[E_f])/2$. We must therefore analyze $\Pr[E_f \land E_g] = \Pr[E_g \mid E_f] \Pr[E_f]$.

The proof would proceed as a sequence of games, where the first is the interaction between the experiment and the adversary.

In the next game, we stop the adversary with a forced guess $b' = 0$ immediately upon any query that would make the execution non-fresh. Note that a query that makes the execution non-fresh can be recognized with no secret information. This changes nothing, but in the further analysis we may assume that the execution remains fresh.

We next simulate all the zero knowledge proofs involved, which is straightforward in the random oracle model since all our proofs are HVZK.

Next, we change the challenge query so that instead of computing the precode as $\hat{r} = a + v$, it samples $\hat{r}$ uniformly at random. If this change is observable, we create a real-or-random adversary against the commitment scheme by making a the challenge and getting a commitment that is either a or a random value.

Next, in the count query, instead of decrypting an encrypted ballot from a challenge query, we use the corresponding left cleartext (regardless of the value of b). Since the shuffle proof has been simulated, the execution is fresh and the ballots are permuted randomly, this change is not observable.

Next, in the count query, instead of decrypting an encrypted ballot from a challenge query, we instead compute the ballot from the corresponding pre-code $\hat{r}$ and voter casting key component a as $v = \hat{r} - a$. This change is only observable if the linear proof verified in the code algorithm was unsound, that is, if we have openings of $\boldsymbol{c}$ to v, of $\boldsymbol{c}_a$ to a, of $\boldsymbol{c} + \boldsymbol{c}_a$ to $v + a$ and $\boldsymbol{c}_{\hat{r}}$ to $\hat{r}$, but $\hat{r} \neq v + a$. Also, the change is observable if the encrypted ballot does not decrypt to an opening, but since the proof for the encrypted ballot was verified during the code query, it follows that this results in an adversary against the verifiability of the encryption scheme.

Observe that at this point, the decryption key $\mathtt{dk}_V$ is no longer used. Also, the pre-code encrypted in the challenge query is independent of the challenge ballots.

In the next game, we encrypt randomness instead of the correct opening of $\boldsymbol{c}$. If this change is observable, we get a real-or-random adversary against the verifiable encryption.

Finally, we commit to a random value instead of the challenge ballot. If this change is observable, we get a real-or-random adversary against the commitment scheme.

We can now observe that the challenge query processing is independent of the challenge bit. The adversary no longer has any information about the challenge bit, and therefore has no advantage in this game. The claim that the difference between $\Pr[E_f \wedge E_g] + \Pr[\neg E_f \wedge E_r]$ and $1/2$ is appropriately bounded follows.

Integrity events Next, we analyze the integrity events. In this case, the adversary may have revealed every secret key, and there is no need for the execution to be fresh.

If a chosen ciphertext query results in an incorrect pre-code, then as above either the ciphertext $\boldsymbol{e}$ does not decrypt to an opening of the commitment (in which case we get an adversary against the verifiability of the encryption scheme), or we have broken the linearity proof for commitments. It follows that the probability of F_i happening is appropriately bounded.

In the event that F_c happens, then since every encrypted ballot either originates with a challenge query or a chosen ciphertext query, we know that either the ciphertext $\boldsymbol{e}$ will decrypt to an opening of the commitment or we will have an adversary against the verifiability of the encryption scheme. In either case, it follows that the probability of F_c happening is appropriately bounded.

In the event that F_d happens, we have openings of the ciphertexts that originated with challenge queries. Since the shuffle is a proof of knowledge, we get an adversary against binding for the commitment scheme. It follows that the probability of F_d happening is appropriately bounded.

In the event that F_u happens, then since the shuffle is a proof of knowledge, we get an adversary against binding for the commitment scheme. It follows that the probability of F_u happening is appropriately bounded.

The claim that $\Pr[F_i \vee F_c \vee F_d \vee F_u]$ is appropriately bounded follows.

B.6 Voting System Security Properties

Coercion Resistance A coercer controls the voter during ballot casting. The appropriate mental image we should have in mind is that the coercer is a "helpful" person who observes the voter while the voter is casting a ballot of the coercer's choice. The coercer may "assist" the voter in the casting process. When the ballot casting is done, the coercer leaves and the voter is left to their own devices. (The coercer may return at a later point in time and redo the coercion process.)

The voter may choose to *resist* the coercion attempt. This may involve active steps during the coerced ballot casting (e.g. lying about authentication data). For remote voting, it will usually also involve *recovery* after the coercer has left.

The coercer plays a game with a set of voters, some of which may be corrupt. The coercer may coerce one or more voters, who will either all resist or all accept coercion. The coercer may also ask voters to cast ballots uncoerced. Eventually, there is a tally and the coercer receives the outcome (including any transcripts).

A system is *coercion resistant* if the coercer cannot decide if the voters resisted.

It is generally assumed that a coercer does not control any infrastructure players and cannot monitor networks, since for remote voting either capability usually allows trivial winning strategies for the coercer. In the real world, a coercer will always be able deduce some information about the success of the coercion attempt by looking at unavoidable public information such as the election result. This applies, regardless of the voting system used (so-called Italian attacks are an example of this). It is desirable to avoid this class of attacks, which can be done by having the coercer decide the voting intention of every voter. The consequence is that the coercer must organize coercion such that it has no effect on the election result, regardless of whether voters resist or not.

Analysis. Our voting system uses re-voting to resist coercion: after the coercer has left, the voter casts another ballot, this time according to their true voting intention. The use of re-voting for coercion resistance is well-understood and largely independent of the underlying cryptosystem. Since the coercer does not learn the contents of the ballot box (except possibly for the commitments), nor any network traffic, it is impossible for the coercer to discover the re-voting.

In summary, the coercer cannot decide if the voter re-voted or not.

Privacy Privacy is modeled as an indistinguishability game between an adversary and a set of voters, some of which may be corrupt. The adversary gives pairs of ballots to honest voters, and they will all either cast the left ballot or the right ballot. The adversary must decide which they cast.

Unlike for coercion resistance, the adversary can corrupt infrastructure players and also control the network. Like for coercion resistance, we want to avoid adversaries that deduce the honest voters' ballots from the cast ballots. Again, we require that the adversary organizes the pairs of ballots given to the honest voters in such a way that the ballots cast are independent of whether the voters cast the left or the right ballot. The difference between privacy and coercion is that for privacy, the adversary learns the ballots cast by compromising infrastructure players. For coercion, the coercer only has access to public information from the infrastructure players, but may have access to private information about the voter.

The adversary controls the network, but we shall assume that players use secure channels to communicate. This means that only the fact that players are communicating and the length of their communications leak. Since message flows and message lengths are fixed and public knowledge, we can ignore the network in the subsequent analysis.

Analysis. If some honest voter's *computer* D is compromised, the adversary can trivially win the privacy game.

Next, consider the case that the *shuffler* S is compromised. If any of B, R or A is compromised, the adversary can trivially win the game, since B, R and A all know who submitted which encrypted ballot, while the shuffler S has the decryption key. If the shuffler S is the only compromised player, then since the ballot box B sorts the list of encrypted ballots before sending them to the shuffler and encrypted ballots are independent of the per-voter key material, the correspondence between the ciphertexts and the voters is lost, so the encrypted ballots alone reveal only the cast ballots. By assumption, these ballots are independent of the left-or-right choice of the voters.

If a voter casts more than one ballot, a compromised *return code generator* will always be able to decide if they are the same or not by observing the return code sent to the voter. If the ballots are distinct (*e.g.* if the voter is resisting coercion), the return code generator will get information about which ballots were submitted, and typically learn both ballots.

Suppose the honest voters cast at most one ballot each. Then privacy against B, R and A follows from confidentiality of the cryptographic voting system, since the protocol execution can be interpreted as an interaction with the cryptosystem experiment and our assumptions ensure a fresh execution.

Note that cut-and-paste attacks against confidentiality, which commonly affect this type of voting protocol, do not work against this protocol because the ballot proof includes an encryption of the return code and a proof that the return code is correct, which means that the adversary must know the ballot to make a cut-and-paste attack work.

In summary, privacy holds if none of the honest voters' computers are compromised, and either only the decryption service is corrupted or no honest voter casts more than one ballot.

Integrity. Integrity for a voting system is modelled using a game between an adversary and a set of voters, some of which may be corrupt. The adversary tells the honest voters what ballots to cast. If the count phase eventually runs and ends with a result, the adversary wins if the result is inconsistent with the ballots accepted as cast by the honest voters. (Recall that only the voter's last ballot cast is counted, so if the voter first accepts a ballot as cast, and then tries to cast another ballot and this fails, the end result is that they have not accepted a ballot as cast.)

We can define a variant notion called ϵ-integrity where we allow a small error, and say that the adversary wins if the result is inconsistent with any $(1 - \epsilon)$ fraction of the ballots accepted as cast by the honest voters. (We need this since return codes for a single voter must be human-comparable, and can therefore collide with some non-negligible probability.)

Analysis. The voter will only accept the ballot as cast if the correct return code is received. If the correct return code is received, then the correct pre-code must have been computed at some point (except with some small probability of collision in the PRF).

If the *return code generator* R is honest, integrity of the cryptographic voting scheme implies that this can only happen if the correct ballot has been encrypted. If the auditor A is honest, the count will only be accepted if the encrypted ballot has been included in the count by the shuffler S. By the integrity of the cryptographic voting system, all such ballots must then be included in the count.

If the voter's computer D, the ballot box B and the auditor A are honest, the count will only be accepted if the encrypted ballot has been included in the count by the shuffler S. By the integrity of the cryptographic voting system, all such ballots must then be included in the count.

If a voter receives a return code without casting a ballot, the voter will no longer accept their ballot as cast.

In summary, ϵ-integrity holds if the auditor and either the return code generator, or both the voters' computers and the ballot box are honest.

Limited Verifiability. In a *verifiable voting system* voters receive a *receipt*, the voting system provides an *election proof* in addition to the election outcome, and there is an additional *verification algorithm* that accepts or rejects the proof and a voter's receipt. Roughly speaking, the voting system is *verifiable* if when all the receipts accepted by the honest voters verify as accepted with the election proof, then the outcome is consistent with the honest voters' cast ballots.

We have *limited verifiability* if the same claim holds when certain players are honest during the election.

(Verifiability is a technical property. The practical idea is not that every voter verifies their ballot. But it can be shown that if a sufficiently large and random sample of voters separately accept their receipts together with the ballot proof, then ϵ-integrity holds for the voting system. In our particular system, if a sufficiently random selection of voters finds their commitments on the public list, then with high probability almost all the voters would have found their commitments on the list if they had looked for them.)

Analysis. If the voters' *computers* are honest, then a voter's computer will not accept the ballot as cast unless it receives a commitment that opens to its encrypted ballot. A voter will not accept the result as verified unless the commitment has been made public.

An honest auditor will only make commitments public if both the ballot box and the return code generator agree on the presence of the encrypted ballot, and this ballot was given to the shuffler. By the integrity of the cryptographic voting system, almost all the ballots accepted as cast by honest voters will be among the ballots output by the decryption service.

In summary, for the variant voting system with limited verifiability, ϵ-integrity holds if the voters' computers and the auditor are honest.

Paper III

Verifiable Mix-Nets and Distributed Decryption for Voting from Lattice-Based Assumptions

Diego F. Aranha, Carsten Baum, Kristian Gjøsteen and Tjerand Silde

The manuscript is available at IACR ePrint:
eprint.iacr.org/2022/422.pdf.

Verifiable Mix-Nets and Distributed Decryption for Voting from Lattice-Based Assumptions

Diego F. Aranha[1], Carsten Baum[1], Kristian Gjøsteen[2], and Tjerand Silde[2*]

[1] Aarhus University, Denmark
{dfaranha,cbaum}@cs.au.dk
[2] Norwegian University of Science and Technology, Norway
{kristian.gjosteen,tjerand.silde}@ntnu.no

Abstract. Cryptographic voting protocols have recently seen much interest from practitioners due to their (planned) use in countries such as Estonia, Switzerland and Australia. Many organizations also use Helios for elections. While many efficient protocols exist from discrete log-type assumptions, the situation is less clear for post-quantum alternatives such as lattices. This is because previous voting protocols do not carry over easily due to issues such as noise growth and approximate relations. In particular, this is a problem for tested designs such as verifiable mixing and decryption of ballot ciphertexts.

In this work, we make progress in this direction. We propose a new verifiable secret shuffle for BGV ciphertexts as well as a compatible verifiable distributed decryption protocol. The shuffle is based on an extension of a shuffle of commitments to known values which is combined with an amortized proof of correct re-randomization. The verifiable distributed decryption protocol uses noise drowning for BGV decryption, proving correctness of decryption steps in zero-knowledge.

We give concrete parameters for our system, estimate the size of each component and provide an implementation of all sub-protocols. Together, the shuffle and the decryption protocol are suitable for use in real-world cryptographic voting schemes, which we demonstrate with a prototype voting protocol design.

Keywords: lattice cryptography · verifiable mix-nets · distributed decryption · zero-knowledge proofs · cryptographic voting · implementation

1 Introduction

Mix-nets were originally proposed for anonymous communication [Cha81], but have since been used extensively for cryptographic voting systems. A mix-net is a multi-party protocol which gets as input a collection of ciphertexts and outputs another collection of ciphertexts whose decryption is the same set, up to order.

* Work done in part while visiting Aarhus University.

The mix-net will mix the ciphertexts so that the permutation between input and output ciphertexts is hidden if at least one party is honest.

Mix-nets are commonly used in cryptographic voting. Here, encrypted ballots are submitted to a bulletin board or ballot box with identifying information attached. These ciphertexts must then be sent through a mix-net before decryption, to break the identity-ballot correlation.

In addition to breaking the correlation between input and output ciphertexts, the correctness of the mix-net must be verifiable. In some cases, it is sufficient that some auditor (possibly distributed) operating at the same time as the mix-net can verify correctness, but for other applications the mix-net should provide a proof of correctness that can be verified by anyone at any later point in time.

A *shuffle* of a set of ciphertexts is another set of ciphertexts whose decryption is the same as the original set, up to order. A shuffle is *secret* if it is hard to correlate input and output ciphertexts. A shuffle is *verifiable* if there is some proof for the claim that the decryptions are the same.

If we have a verifiable secret shuffle for some cryptosystem, building a mix-net is trivial. The nodes of the mix-net receive a set of ciphertexts as input, shuffle them sequentially and provide a proof of correctness. The mix-net proof then consists of the sets of intermediate ciphertexts along with the shuffle proofs. If at least one node in the mix-net is honest, it is hard to correlate the inputs and outputs.

For applications in cryptographic voting, we also need verifiable decryption to ensure that the correct result can be obtained. The design of the voting system must ensure that nobody has both the decryption key and the original ciphertexts. One simple strategy for this is to use verifiable threshold decryption, where the decryption key is secret-shared among a committee of decryption parties.

Verifiable shuffling and verifiable distributed decryption protocols are well-known for cryptosystems based on discrete log-type assumptions. For example, Neff [Nef01] proposed the first efficient verifiable secret shuffle for ElGamal-like cryptosystems. Distributed verifiable decryption can be achieved by giving uniformly random shares of the secret key to a set decryption nodes, and have each of them compute a partial decryption together with a proof of equality of discrete logarithms [CP93] with respect to the secret key share.

Quantum-safety is critical for cryptographic voting systems, since elections have a long-term need for privacy, and there is an urgent need for progress. Verifiable secret shuffles and verifiable distributed decryption are two long-standing obstacles to delivering practical cryptographic voting schemes based on quantum-safe computational problems such as lattice assumptions.

1.1 Our contributions

In this paper, we design a verifiable secret shuffle for BGV ciphertexts [BGV12] that is suitable for cryptographic voting systems. The main obstacle to simply adopting the ideas used by Neff for discrete logarithms to lattices is the lack of suitable underlying proofs and techniques. We overcome these obstacles by designing an extended version of the shuffle of commitments to known values by Aranha *et al.* [ABG+21]. In our protocol, the shuffler gets input ciphertexts $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau$. We let the shuffler commit to re-randomization ciphertexts $\hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_\tau$ using a suitable linearly homomorphic commitment scheme $\texttt{Com}$. Together with an amortized proof of shortness of the randomness used for the committed re-randomization ciphertexts, this gives us a verifiable shuffle:

1. First, the shuffler commits to the re-randomization ciphertexts $\hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_\tau$ as $\texttt{Com}(\hat{\boldsymbol{c}}_i)$ and shows that they are well-formed.
2. The shuffler computes $\boldsymbol{d}_i = \boldsymbol{c}_i + \hat{\boldsymbol{c}}_i$ and sends shuffled elements $L = (\boldsymbol{d}_{\pi(i)})_{i \in [\tau]}$ to the receiver.
3. Finally, the prover shows that L is a list of openings of the commitments obtained from $\boldsymbol{c}_i + \texttt{Com}(\hat{\boldsymbol{c}}_i)$.

Towards implementing this, we use highly efficient lattice-based commitments [BDL+18] together with a version of recent amortized proofs of shortness [BLNS21].

As explained, a verifiable secret shuffle on its own is usually not enough to build a cryptographic voting system. We also need some way to decrypt the output of the mix-net, without compromising the input ciphertexts or allowing a decryption server to cheat. Our solution is to distribute the decryption operation in a verifiable way. We hand out key-shares of the secret decryption key to each decryption server, and all of them perform a partial decryption of each ciphertext. In addition, we publish commitments to the key shares. The decryption servers then add noise to the partial decryption to hide information about their shares, called noise drowning [BD10]. Finally they publish the partial decryptions together with a proof of correctness of the decryption (and boundedness of the noise used), and the plaintexts are computed in public by combining all the partial decryptions.

Lattice-based cryptography is very delicate with respect to noise-levels, bounds and dimensions, and we have to be cautious when combining the sub-protocols mentioned into a larger construction. Each shuffle adds extra noise to each ciphertext, which means that to ensure correctness of decryption we need to choose specific parameters based on the number of shuffles and the norm of the noise added in each shuffle. Here, the norm is guaranteed by the zero-knowledge proofs of shortness accompanying each shuffle. Furthermore, each partial decryption also adds noise to the ciphertexts to hide the secret key. Because of the noise drowning technique, the norm must be quite large, influencing both the bounds of the amortized zero-knowledge proof and the choice of parameters for the overall cryptosystem. In particular, it is important when measuring performance to use parameters suitable for the complete system, not parameters optimized for individual components only.

In order to provide proper context for our contributions, we provide a sketch of a full cryptographic voting protocol. A simplified variant could be used as a quantum-safe Helios [Adi08] variant. We give example parameters suitable for this protocol with 4 mix-nodes and 4 decryption nodes. We have estimated the size of each component with respect to the parameters for the full protocol in addition to implementing all sub-protocols, showing that it can be used for large-scale real-world elections where ballots typically are counted and verified in batches of tens of thousands.

To summarize our implementation results, a ciphertext ballot is of size 80 KB, each mixing proof is of total size 370τ KB and each decryption proof is of total size 157τ KB, where τ is the number of total ciphertexts. It takes only 2.5 ms to encrypt a ballot, while the mixing proof takes 1024τ ms and the decryption proof takes 81.4τ ms. Note that the shuffle proof only takes 15.1τ ms, so the time it takes to mix the ciphertexts is dominated by the time it takes to prove correct re-randomization, which is 1009τ ms. Verification is much faster, only 20τ ms. These results improve on the state of the art considerably, see details in Section 7.

1.2 Related work

Aranha *et al.* [ABG+21] provide a verifiable shuffle of known commitment openings together with concrete parameters and an implementation of a complete voting protocol. However, their trust model has the limitation that the ballot box and the shuffle server must not collude to ensure privacy of the ballots, which is too restrictive for most real-world settings. This is inherent for the protocol which can not easily be extended to several shuffles unless layered encryption is used, and this would heavily impact the performance.

Costa *et al.* [CMM19] design a more general shuffle with a straight-forward approach similar to Neff [Nef01] based on roots of polynomials. Their protocol requires committing to two evaluations of a polynomial, and then prove the correctness of evaluation using a sequence of multiplication proofs which are quite costly in practice. Farzaliyev *et al.* [FWK21] implements the mix-net by Costa *et al.* [CMM19] using the amortization techniques by Attema *et al.* [ALS20] for the commitment scheme by Baum *et al* [BDL+18]. Here, the proof size is approximately 15 MB per voter, a factor 40 larger than our shuffle proof, even for a smaller parameter set that does not take into account distributed decryption afterwards. We expect our shuffle proof to be an additional factor 10 smaller than what we presented above with optimal parameters for the shuffle only ($q \approx 2^{32}$ and $N = 1024$). Furthermore, their proof generation takes approximately 1.5 seconds per vote, which is approximately 40 % faster than it takes to produce our shuffle proof (when normalizing for clock frequency), with parameters that do not take decryption into account.

Recently, Herranz *et al.* [HMS21] gave a new proof of correct shuffle based on Benes networks and sub-linear lattice-based proofs for arithmetic circuit satisfiability. However, the scheme is not implemented and the example parameters do not take the soundness slack of the amortized zero-knowledge proofs into account. Moreover, [HMS21] does not consider decryption of ballots, which would heavily impact the parameters of their protocol in practice.

A completely different approach to mix-nets is so-called decryption mix-nets. The idea is that the input ciphertexts are actually nested encryptions. Each node in the mix-net is then responsible for decrypting one layer of each ciphertext. These can be made fully generic, relying only on public key encryption. Boyen *et al.* [BHM20] carefully adapt these ideas to lattice-based encryption, resulting in a very fast scheme. Decryption mix-nets are well-suited to applications in anonymous communication. However, for voting applications they are often less well-suited due to their trust requirements. An important goal for cryptographic voting is universal verifiability: after the election is done, anyone should be able to verify that the ballot decryption was done correctly without needing to trust anyone. This trust issue generalizes to any situation where it is necessary to convince someone that a shuffle has been done, but no auditor is available. Fast or generic decryption mix-nets such as Boyen *et al.* [BHM20] need an auditor (that can be distributed) to verify the mix-net, but the auditor must be trusted during the mix-net operation. This conflicts with universal verifiability.

del Pino *et al.* [dLNS17] give a practical voting protocol based on homomorphic counting. They only support yes/no-elections, and the total size depends directly on the number of candidates for larger elections. It was shown by Boyen *et al.* [BHM21] that the protocol in [dLNS17] is not end-to-end verifiable unless all tallying authorities and all voters' voting devices are honest. This problem is solved by [BHM21], but their construction still has the downside of only supporting homomorphic tallying. Strand [Str19] built a verifiable shuffle for the GSW cryptosystem, but this construction is too restrictive for practical use. Chillotti *et al.* [CGGI16] uses fully homomorphic encryption, which for the foreseeable future is most likely not efficient enough to be considered for practical deployment.

2 Preliminaries

Let N be a power of 2 and q a prime such that $q \equiv 1 \mod 2N$. We define the rings $R = \mathbb{Z}[X]/\langle X^N + 1\rangle$ and $R_q = R/qR$, that is, R_q is the ring of polynomials modulo $X^N + 1$ with integer coefficients modulo q. This way, $X^N + 1$ splits completely into N irreducible factors modulo q, which allows for very efficient computation in R_q due to the number theoretic transform (NTT) [LN16]. We define the norms of elements $f(X) = \sum \alpha_i X^i \in R$ to be the norms of the coefficient vector as a vector in $\mathbb{Z}^N$:

$$||f||_1 = \sum |\alpha_i|, \qquad ||f||_2 = \left(\sum \alpha_i^2\right)^{1/2}, \qquad ||f||_\infty = \max_{i\in[1,\dots,n]} \{|\alpha_i|\}.$$

For an element $\bar{f} \in R_q$ we choose coefficients as the representatives in $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$, and then compute the norms as if $\bar{f}$ is an element in R. For vectors $\boldsymbol{a} = (a_1, \dots, a_k) \in R^k$ we define the ℓ_2 norm to be $\|\boldsymbol{a}\|_2 = \sqrt{\sum \|a_i\|_2^2}$, and analogously for the ℓ_1 and ℓ_∞ norm. It is easy to see the following relations between the norms of elements in R_q:

$$\|f\|_\infty \leq \alpha, \|g\|_1 \leq \beta, \text{ then } \|fg\|_\infty \leq \alpha\beta,$$
$$\|f\|_2 \leq \alpha, \|g\|_2 \leq \beta, \text{ then } \|fg\|_\infty \leq \alpha\beta.$$

We furthermore define the sets $S_{\beta_\infty} = \{x \in R_q \mid \|x\|_\infty \leq \beta_\infty\}$ as well as

$$\mathcal{C} = \{c \in R_q \mid \|c\|_\infty = 1, \|c\|_1 = \nu\} \text{ and } \bar{\mathcal{C}} = \{c - c' \mid c \neq c' \in \mathcal{C}\}.$$

2.1 The Discrete Gaussian Distribution

The continuous normal distribution over $\mathbb{R}^k$ centered at $\boldsymbol{v} \in \mathbb{R}^k$ with standard deviation σ is given by

$$\rho^N_{\boldsymbol{v},\sigma}(\boldsymbol{x}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-||\boldsymbol{x} - \boldsymbol{v}||^2}{2\sigma^2} \right).$$

When sampling randomness for our lattice-based commitment and encryption schemes, we will need samples from the *discrete Gaussian distribution.* This distribution is achieved by normalizing the continuous distribution over R^k by letting

$$\mathcal{N}^k_{\boldsymbol{v},\sigma}(\boldsymbol{x}) = \frac{\rho^{kN}_{\boldsymbol{v},\sigma}(\boldsymbol{x})}{\rho^{kN}_{\sigma}(R^k)} \text{ where } \boldsymbol{x} \in R^k \text{ and } \rho^{kN}_{\sigma}(R^k) = \sum_{\boldsymbol{x} \in R^k} \rho^{kN}_{\sigma}(\boldsymbol{x}).$$

When $\sigma = 1$ or $\boldsymbol{v} = \boldsymbol{0}$, they are omitted. When $\boldsymbol{x}$ is sampled according to $\mathcal{N}_\sigma$ (see Section 2.1 in [BBC+18]), then,

$$\texttt{Pr}[\|\boldsymbol{x}\|_\infty > \gamma\sigma] \leq 2e^{-\gamma^2/2} \quad \text{and} \quad \texttt{Pr}[\|\boldsymbol{x}\|_2 > \sqrt{2\gamma}\sigma] < 2^{-\gamma/4}.$$

2.2 Rejection Sampling

In lattice-based cryptography in general, and in our zero-knowledge protocols in particular, we would like to output vectors $\boldsymbol{z} = \boldsymbol{y} + \boldsymbol{v}$ such that $\boldsymbol{z}$ is independent of $\boldsymbol{v}$, and hence, $\boldsymbol{v}$ is masked by the vector $\boldsymbol{y}$. Here, $\boldsymbol{y}$ is sampled according to a Gaussian distribution $\mathcal{N}^k_\sigma$ with standard deviation σ, and we want the output vector $\boldsymbol{z}$ to be from the same distribution. The procedure is shown in Figure 1.

Here, $1/M$ is the probability of success, and M is computed as

$$\max \frac{\mathcal{N}^k_\sigma(\boldsymbol{z})}{\mathcal{N}^k_{\boldsymbol{v},\sigma}(\boldsymbol{z})} = \exp\left[\frac{-2\langle \boldsymbol{z}, \boldsymbol{v} \rangle + \|\boldsymbol{v}\|_2^2}{2\sigma^2} \right] \leq \exp\left[\frac{24\sigma\|\boldsymbol{v}\|_2 + \|\boldsymbol{v}\|_2^2}{2\sigma^2} \right] = M, \quad (1)$$

where we use the tail bound from Section 2.1, saying that $|\langle \boldsymbol{z}, \boldsymbol{v} \rangle| < 12\sigma\|\boldsymbol{v}\|_2$ with probability at least $1 - 2^{100}$. Hence, for $\sigma = 11\|\boldsymbol{v}\|_2$, we get $M \approx 3$. This is the standard way to choose parameters, see e.g. [BLS19]. However, if the procedure is only done once for the vector $\boldsymbol{v}$, we can decease the parameters slightly, to the cost of leaking only one bit of information about $\boldsymbol{v}$ from the given $\boldsymbol{z}$.

In [LNS21], Lyubashevsky *et al.* suggest to require that $\langle \boldsymbol{z}, \boldsymbol{v} \rangle \geq 0$, and hence, we can set $M = \exp(\|v\|_2/2\sigma^2)$. Then, for $\sigma = 0.675\|\boldsymbol{v}\|_2$, we get $M \approx 3$. In Figure 1, we use the pre-determined bit b to denote if we only use $\boldsymbol{v}$ once or not, with the effect of rejecting about half of the vectors before the sampling of uniform value u in the case $b = 1$, but allowing a smaller standard deviation.

```
Rej(z, v, b, M, σ)
1 : if $b = 1$ and $\langle \boldsymbol{z}, \boldsymbol{v} \rangle < 0$: return 1
2 : $\mu \leftarrow\!\!\$\ [0, 1)$
3 : if $\mu > \frac{1}{M} \cdot \exp\left[\frac{-2\langle \boldsymbol{z}, \boldsymbol{v} \rangle + \|\boldsymbol{v}\|_2^2}{2\sigma^2}\right]$ : return 1
4 : else : return 0
```

Fig. 1. Rejection sampling

2.3 Knapsack Problems

We first define the Search Knapsack problem in the ℓ_2 norm, also denoted as SKS^2. The SKS^2 problem is exactly the Module-SIS problem in its Hermite Normal Form.

Definition 1 (Search Knapsack problem). *The $\mathsf{SKS}^2_{n,k,\beta}$ problem is to find a short non-zero vector $\boldsymbol{y}$ satisfying $[\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y} = \boldsymbol{0}^n$ for a random matrix $\boldsymbol{A}'$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the $\mathsf{SKS}^2_{n,k,\beta}$ problem if*

$$\Pr\left[\begin{array}{c|c} \|y_i\|_2 \leq \beta \wedge & \boldsymbol{A}' \leftarrow\!\!\$\ R_q^{n\times(k-n)}; \\ [\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y} = \boldsymbol{0}^n & \boldsymbol{0} \neq \boldsymbol{y} = [y_1, \ldots, y_k]^\top \leftarrow \mathcal{A}(\boldsymbol{A}') \end{array}\right] \geq \epsilon.$$

Additionally, we define the Decisional Knapsack problem in the ℓ_∞ norm (DKS^∞). The DKS^∞ problem is equivalent to the Module-LWE problem when the number of samples is limited.

Definition 2 (Decisional Knapsack problem). *The $\mathsf{DKS}^\infty_{n,k,\beta}$ problem is to distinguish the distribution $[\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y}$ for a short $\boldsymbol{y}$ from a bounded distribution S_{β_∞} when given $\boldsymbol{A}'$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the $\mathsf{DKS}^\infty_{n,k,\beta}$ problem if*

$$\Big|\Pr[b = 1 \mid \boldsymbol{A}' \leftarrow\!\!\$\ R_q^{n\times(k-n)}; \boldsymbol{y} \leftarrow\!\!\$\ S_{\beta_\infty}^k; b \leftarrow \mathcal{A}(\boldsymbol{A}', [\ \boldsymbol{I}_n \quad \boldsymbol{A}'\ ] \cdot \boldsymbol{y})]$$
$$- \Pr[b = 1 \mid \boldsymbol{A}' \leftarrow\!\!\$\ R_q^{n\times(k-n)}; \boldsymbol{u} \leftarrow\!\!\$\ R_q^n; b \leftarrow \mathcal{A}(\boldsymbol{A}', \boldsymbol{u})]\Big| \geq \epsilon.$$

See [LS15] for more details about hardness problems over module lattices.

2.4 Public Key Encryption

We present definitions inspired by Goldwasser and Micali [GM82] for the security of a (slightly additively homomorphic) public key encryption scheme. We only present chosen plaintext (CPA) security here, as we need to randomize ciphertexts in our main protocol. To ensure full security one often requires chosen ciphertext security, which can be achieved by combining a CPA secure scheme with zero-knowledge proofs of correct encryption.

Definition 3 (Public Key Encryption Scheme). *A public key encryption scheme consists of three algorithms: key generation* (KeyGen)*, encryption* (Enc) *and decryption* (Dec)*, where*

- KeyGen*, on input security parameter* 1^λ*, outputs public parameters* pp*, a public key* pk*, and a secret key* sk*,*
- Enc*, on input the public key* pk *and a message* m*, outputs a ciphertext* c*,*
- Dec*, on input the secret key* sk *and a ciphertext* c*, outputs a message* m*,*

and the public parameters pp *are implicit inputs to* Enc *and* Dec*.*

Definition 4 (τ-Correctness). *We say that the public key encryption scheme is* τ-correct *if a sum of* τ *honestly generated ciphertext with overwhelming probability decrypts to the sum of the* τ *encrypted messages. Hence, we want that*

$$\Pr\left[\mathtt{Dec}(\mathtt{sk}, \sum_{i\in[\tau]} c_i) = \sum_{i\in[\tau]} m_i \;:\; \begin{array}{c} (\mathtt{pp}, \mathtt{pk}, \mathtt{sk}) \leftarrow \mathtt{KeyGen}(1^\lambda) \\ \{c_i\}_{i\in[\tau]} \leftarrow \mathtt{Enc}(\mathtt{pk}, \{m_i\}_{i\in[\tau]}) \end{array}\right] \geq 1 - \epsilon(\lambda),$$

where the probability is taken over the random coins of KeyGen *and* Enc*.*

Definition 5 (Chosen Plaintext Security). *We say that the public key encryption scheme is secure against* chosen plaintext attacks *if an adversary* $\mathcal{A}$*, after choosing two messages* m_0 *and* m_1 *and receiving an encryption* c *of either* m_0 *or* m_1 *(chosen at random), cannot distinguish which message* c *is an encryption of. Hence, we want that*

$$|\Pr\left[b = b' \;:\; \begin{array}{c} (\mathtt{pp}, \mathtt{pk}, \mathtt{sk}) \leftarrow \mathtt{KeyGen}(1^\lambda) \\ (m_0, m_1, \mathtt{st}) \leftarrow \mathcal{A}(\mathtt{pp}, \mathtt{pk}) \\ b \xleftarrow{\$} \{0,1\}, c \leftarrow \mathtt{Enc}(\mathtt{pk}, m_b) \\ b' \leftarrow \mathcal{A}(c, \mathtt{st}) \end{array}\right] - \frac{1}{2}| \leq \epsilon(\lambda),$$

where the probability is taken over the random coins of KeyGen *and* Enc*.*

2.5 Public Key Distributed Decryption

We now present a definition of a secure public key distributed decryption protocol that is suitable for our voting application.

Definition 6 (Distributed Decryption Scheme). *A public key distributed decryption scheme consists of five algorithms: key generation* (KeyGen)*, encryption* (Enc)*, decryption* (Dec)*, distributed decryption* (DistDec) *and combine* (Comb)*, where*

- KeyGen*, on input security parameter* 1^λ *and number of key-shares* ξ*, outputs public parameters* pp*, a public key* pk*, a secret key* sk*, and key-shares* $\{\mathtt{sk}_j\}$*,*
- Enc*, on input the public key* pk *and messages* $\{m_i\}$*, outputs ciphertexts* $\{c_i\}$*,*
- Dec*, on input the secret key* sk *and ciphertexts* $\{c_i\}$*, outputs messages* $\{m_i\}$*,*
- DistDec*, on input a secret key-share* $\mathtt{sk}_{j^*}$ *and ciphertexts* $\{c_i\}$*, outputs decryption-shares* $\{\mathtt{ds}_{i,j^*}\}$*,*
- Comb*, on input ciphertexts* $\{c_i\}$ *and decryption-shares* $\{\mathtt{ds}_{i,j}\}$*, outputs either messages* $\{m_i\}$ *or* $\perp$*,*

and the public parameters pp *are implicit inputs to* Enc*,* Dec*,* DistDec *and* Comb*.*

Let $P_{\mathsf{sk}}(u, v)$ be an efficiently computable predicate that on input secret key $\mathsf{sk} = s$ and a ciphertext $c = (u, v)$ outputs 1 or 0. For example, it outputs 1 if $\|v - su\|_\infty < B_{\mathtt{Dec}} \ll \lfloor q/2 \rfloor$ and otherwise 0 for the BGV scheme in Section 3.1.

Definition 7 (Threshold Correctness). *We say that the public key distributed encryption scheme is* threshold correct *with respect to to* $P_{\mathsf{sk}}(\cdot)$ *if*

$$\Pr\left[\begin{array}{c} \mathtt{Comb}(\{c_i\}_{i\in[\tau]}, \{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\in[\xi]}) \\ = \\ \mathtt{Dec}(\mathsf{sk}, \{c_i\}_{i\in[\tau]}) \end{array} : \begin{array}{c} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, \{\mathsf{sk}_j\}_{j\in[\xi]}) \leftarrow \mathtt{KeyGen}(1^\lambda, \xi) \\ \{c_1, \ldots, c_\tau\} \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}) \\ \forall i \in [\tau] : \ P_{\mathsf{sk}}(c_i) = 1 \\ \forall j \in [\xi] : \ \{\mathtt{ds}_{i,j}\}_{i\in[\tau]} \leftarrow \mathtt{DistDec}(\mathsf{sk}_j, \{c_i\}_{i\in[\tau]}) \end{array}\right] = 1,$$

where the probability is taken over the random coins of KeyGen *and* DistDec*.*

Definition 8 (Threshold Verifiability). *We say that the public key distributed encryption scheme is* threshold verifiable *with respect to* $P_{\mathsf{sk}}(\cdot)$ *if an adversary* $\mathcal{A}$ *corrupting* $J \subseteq [\xi]$ *secret key-shares* $\{\mathtt{sk}_j\}_{j\in J}$ *cannot convince* Comb *to accept maliciously created decryption-shares* $\{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\in J}$*. More concretely:*

$$\Pr\left[\begin{array}{c} \mathtt{Dec}(\mathsf{sk}, \{c_i\}_{i\in[\tau]}) \\ \neq \\ \mathtt{Comb}(\{c_i\}_{i\in[\tau]}, \{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\in[\xi]}) \\ \neq \\ \bot \end{array} : \begin{array}{c} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, \{\mathsf{sk}_j\}_{j\in[\xi]}) \leftarrow \mathtt{KeyGen}(1^\lambda, \xi) \\ (\{c_1, \ldots, c_\tau\}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \{\mathsf{sk}_j\}_{j\in J}) \\ \forall i \in [\tau] : \ P_{\mathsf{sk}}(c_i) = 1 \\ \forall j \notin J : \{\mathtt{ds}_{i,j}\}_{i\in[\tau]} \leftarrow \mathtt{DistDec}(\mathsf{sk}_j, \{c_i\}_{i\in[\tau]}) \\ \{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\in J} \leftarrow \mathcal{A}(\{\mathtt{ds}_{i,j}\}_{i\in[\tau],j\notin J}, \mathsf{st}) \end{array}\right] \leq \epsilon(\lambda),$$

where the probability is taken over the random coins of KeyGen *and* DistDec*.*

Definition 9 (Distributed Decryption Simulatability). *We say that the public key distributed decryption scheme is* simulatable *with respect to* $P_{\mathsf{sk}}(\cdot)$ *if an adversary* $\mathcal{A}$ *corrupting* $J \subsetneq [\xi]$ *secret key-shares* $\{\mathtt{sk}_j\}_{j\in J}$ *cannot distinguish the transcript of the decryption protocol from a simulation by a simulator* $\mathcal{S}$ *which only gets* $\{\mathtt{sk}_j\}_{j\in J}$ *as well as correct decryptions as input. More concretely:*

$$\left|\Pr\left[b = b' \ : \begin{array}{c} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, \{\mathsf{sk}\}_{j\in[\xi]}) \leftarrow \mathtt{KeyGen}(1^\lambda, \xi) \\ (\{c_1, \ldots, c_\tau\}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \{\mathsf{sk}_j\}_{j\in J}) \\ \forall i \in [\tau] : \ P_{\mathsf{sk}}(c_i) = 1 \\ \{\mathtt{ds}^0_{i,j}\} \leftarrow \mathtt{DistDec}(\{\mathsf{sk}_j\}_{j\in[\xi]}, \{c_i\}_{i\in[\tau]}) \\ \{\mathtt{ds}^1_{i,j}\} \leftarrow \mathcal{S}(\mathsf{pp}, \{\mathsf{sk}_j\}_{j\in J}, \{c_i, \mathtt{Dec}(\mathsf{sk}, c_i)\}_{i\in[\tau]}) \\ b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathcal{A}(\{\mathtt{ds}^b_{i,j}\}_{i\in[\tau],j\in[\xi]}, \mathsf{st}) \end{array}\right] - \frac{1}{2}\right| \leq \epsilon(\lambda),$$

where the probability is taken over the random coins of KeyGen, DistDec *and* $\mathcal{S}$*.*

2.6 Commitments

Commitment schemes were first introduced by Blum [Blu84], and have since become an essential component in many advanced cryptography protocols.

Definition 10 (Commitment Scheme). *A commitment scheme consists of three algorithms: key generation* (KeyGen)*, commitment* (Com) *and opening* (Open)*, where*

- KeyGen*, on input security parameter* 1^λ*, outputs public parameters* pp*,*

- Com, *on input message m, outputs commitment c and opening r,*
- Open, *on input m, c and r, outputs either* 0 *or* 1,

and the public parameters pp *are implicit inputs to* Com *and* Open.

Definition 11 (Completeness). *We say that the commitment scheme is* complete *if an honestly generated commitment is accepted by the opening algorithm. Hence, we want that*

$$\Pr\left[\texttt{Open}(m,c,r)=1 \;:\; \begin{matrix}\texttt{pp}\leftarrow\texttt{KeyGen}(1^\lambda)\\(c,r)\leftarrow\texttt{Com}(m)\end{matrix}\right]=1,$$

where the probability is taken over the random coins of KeyGen *and* Com.

Definition 12 (Hiding). *We say that a commitment scheme is* hiding *if an adversary $\mathcal{A}$, after choosing two messages m_0 and m_1 and receiving a commitment c to either m_0 or m_1 (chosen at random), cannot distinguish which message c is a commitment to. Hence, we want that*

$$|\Pr\left[b=b' \;:\; \begin{matrix}\texttt{pp}\leftarrow\texttt{KeyGen}(1^\lambda)\\(m_0,m_1,\texttt{st})\leftarrow\texttt{A}(\texttt{pp})\\b\xleftarrow{\$}\{0,1\},c\leftarrow\texttt{Com}(m_b)\\b'\leftarrow\texttt{A}(c,\texttt{st})\end{matrix}\right]-\frac{1}{2}|\leq\epsilon(\lambda),$$

where the probability is taken over the random coins of KeyGen *and* Com.

Definition 13 (Binding). *We say that a commitment scheme is* binding *if an adversary $\mathcal{A}$, after creating a commitment c, cannot find two valid openings to c for different messages m and $\hat{m}$. Hence, we want that*

$$\Pr\left[\begin{matrix}m\neq\hat{m}\\\texttt{Open}(m,c,r)=1\\\texttt{Open}(\hat{m},c,\hat{r})=1\end{matrix} \;:\; \begin{matrix}\texttt{pp}\leftarrow\texttt{KeyGen}(1^\lambda)\\(c,m,r,\hat{m},\hat{r})\leftarrow\texttt{A}(\texttt{pp})\end{matrix}\right]\leq\epsilon(\lambda),$$

where the probability is taken over the random coins of KeyGen.

2.7 Zero-Knowledge Proofs

These definitions are based on Goldwasser et al. [GMR85]. Let L be a language, and let R be a NP-relation on L. Then, x is an element in L if there exists a witness w such that $(x,w)\in$ R. We let P, P*, V and V* be polynomial time algorithms.

Definition 14 (Interactive Proofs). *An interactive proof protocol* Π *consists of two parties: a prover* P *and a verifier* V, *and a setup algorithm* (Setup), *where* Setup, *on input the security parameter 1^λ, outputs public setup parameters* sp. *The protocol consists of a transcript* T *of the communication between* P *and* V, *with respect to* sp, *and the conversation terminates with* V *outputting either* 1 *or* 0. *Let* $\langle\texttt{P}(\texttt{sp},x,w),\texttt{V}(\texttt{sp},x)\rangle$ *denote the output of* V *on input x after its interaction with* P, *who holds a witness w.*

Definition 15 (Completeness). *We say that a proof protocol* Π *is* complete *if* $\mathtt{V}$ *outputs* 1 *when* $\mathtt{P}$ *knows a witness* w *and both parties follows the protocol. Hence, for any efficient sampling algorithm* $\mathtt{P}_0$ *we want that*

$$\Pr\left[\langle \mathtt{P}(\mathtt{sp}, x, w), \mathtt{V}(\mathtt{sp}, x)\rangle = 1 \ : \ \begin{array}{c} \mathtt{sp} \leftarrow \mathtt{Setup}(1^\lambda) \\ (x, w) \leftarrow \mathtt{P}_0(\mathtt{sp}) \\ (x, w) \in \mathtt{R} \end{array}\right] = 1,$$

where the probability is taken over the random coins of $\mathtt{Setup}, \mathtt{P}$ *and* $\mathtt{V}$*.*

Definition 16 (Knowledge Soundness). *We say that a proof protocol* Π *is* knowledge sound *if, when a cheating prover* $\mathtt{P}^*$ *that does not know a witness* w *is able to convince a honest verifier* $\mathtt{V}$*, there exists a polynomial time algorithm extractor* $\mathcal{E}$ *which, give black-box access to* $\mathtt{P}^*$*, can output a witness* w *such that* $(x, w) \in \mathtt{R}$*. Hence, we want that*

$$\Pr\left[(x, w) \in \mathtt{R} \ : \ \begin{array}{c} \mathtt{sp} \leftarrow \mathtt{Setup}(1^\lambda) \\ \langle \mathtt{P}^*(\mathtt{sp}, x, \cdot), \mathtt{V}(\mathtt{sp}, x)\rangle = 1 \\ w \leftarrow \mathcal{E}^{\mathtt{P}^*(\cdot)}(\mathtt{sp}, x) \end{array}\right] \geq 1 - \epsilon(\lambda),$$

where the probability is taken over the random coins of $\mathtt{Setup}, \mathtt{P}^*$ *and* $\mathcal{E}$*.*

Definition 17 (Honest-Verifier Zero-Knowledge). *We say that a proof protocol* Π *is* honest-verifier zero-knowledge *if a honest but curious verifier* $\mathtt{V}^*$ *that follows the protocol cannot learn anything beyond the fact that* $x \in \mathtt{L}$*. Hence, we want for real accepting transcripts* $\mathtt{T}_{\langle \mathtt{P}(\mathtt{sp},x,w),\mathtt{V}(\mathtt{sp},x)\rangle}$ *between a prover* $\mathtt{P}$ *and a verifier* $\mathtt{V}$*, and a accepting transcript* $\mathtt{S}_{\langle \mathtt{P}(\mathtt{sp},x,\cdot),\mathtt{V}(\mathtt{sp},x)\rangle}$ *generated by simulator* $\mathcal{S}$ *that only knows* x*, that*

$$|\Pr\left[b = b' \ : \ \begin{array}{c} \mathtt{sp} \leftarrow \mathtt{Setup}(1^\lambda) \\ \mathtt{T}_0 = \mathtt{T}_{\langle \mathtt{P}(\mathtt{sp},x,w),\mathtt{V}(\mathtt{sp},x)\rangle} \leftarrow \Pi(\mathtt{sp}, x, w) \\ \mathtt{T}_1 = \mathtt{S}_{\langle \mathtt{P}(\mathtt{sp},x,\cdot),\mathtt{V}(\mathtt{sp},x)\rangle} \leftarrow \mathcal{S}(\mathtt{sp}, x) \\ b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathtt{V}^*(\mathtt{sp}, x, \mathtt{T}_b) \end{array}\right] - \frac{1}{2}| \leq \epsilon(\lambda),$$

where the probability is taken over the random coins of $\mathtt{Setup}, \mathcal{S}$ *and* $\mathtt{V}^*$*.*

An interactive honest-verifier zero-knowledge proof protocol can be made non-interactive using the Fiat-Shamir transform [FS87].

3 Background: Lattice-Based Cryptography

We start this section by presenting the BGV encryption scheme by Brakerski *et al.* [BGV12], and continue by presenting the commitment scheme by Baum *et al.* [BDL+18] and it's respective zero-knowledge proofs of linear relations. Then, we present two amortized zero-knowledge proofs of knowledge of short preimages. First, we present a modification of the protocol due to Bootle *et al.* [BLNS21] for proving knowledge of many instances when the secrets are very short, say, all coefficients are ternary, and the proof must be exact. We also provide the protocol due to Baum *et al.* [BBC+18] for proving knowledge of many bounded instances where the proof is approximate.

3.1 BGV Encryption

Let $p \ll q$ be primes, let R_q and R_p be defined as above for a fixed N, let $\mathtt{D}$ be a bounded distribution over R_q, let $B_\infty \in \mathbb{N}$ be a bound and let λ be the security parameter. The (plain) BGV encryption scheme follows Definition 3 (in Section 2) and consists of three algorithms: key generation (KeyGen), encryption (Enc) and decryption (Dec), where:

- KeyGen samples an element $a \leftarrow\!\!\$\ R_q$ uniformly at random, samples a short $s \leftarrow\!\!\$\ R_q$ such that $\|s\|_\infty \leq B_\infty$ and samples noise $e \leftarrow \mathtt{D}$. The algorithm outputs public key $\mathsf{pk} = (a, b) = (a, as + pe)$ and secret key $\mathsf{sk} = s$.

- Enc, on input the public key $\mathsf{pk} = (a, b)$ and an element m in R_p, samples a short $r \leftarrow\!\!\$\ R_q$ such that $\|r\|_\infty \leq B_\infty$, samples noise $e', e'' \leftarrow \mathtt{D}$, and outputs the ciphertext $c = (u, v) = (ar + pe', br + pe'' + m)$.

- Dec, on input the secret key $\mathsf{sk} = s$ and a ciphertext $c = (u, v)$ in R_q^2, outputs the message $m = (v - su \mod q) \mod p$.

The following theorem for the BGV encryption scheme follows from Brakerski *et al.* [BGV12, Section 3] and Lyubashevsky *et al.* [LPR13, Lemma 8.3].

Theorem 1 (Correctness and CPA Security of BGV). *The BGV encryption scheme is* 1-correct *(Definition 4 in 2.4) if* $\|v - su\|_\infty \leq B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$*, and the scheme is secure against* chosen plaintext attacks *(Definition 5 in 2.4) if the* $\mathsf{DKS}^\infty_{N,2,\beta}$ *problem is hard for some* $\beta = \beta(N, q, B_\infty, \sigma, p)$*. More general, the scheme is* τ-correct *if* $\tau \cdot B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$*.*

Furthermore, we present the passively secure distributed decryption technique used in the MPC-protocols by Damgård *et al.* [BD10, DKL⁺13, DPSZ12]. Here the KeyGen algorithm for $1 \leq j \leq \xi$ outputs uniformly random shares $\mathsf{sk}_j = s_j$ of the secret key $\mathsf{sk} = s$ such that $s = s_1 + s_2 + \cdots + s_\xi$. This can be used to define a passively secure threshold decryption algorithm as follows:

- DistDec, on input a secret key-share $\mathsf{sk}_j = s_j$ and a ciphertext $c = (u, v)$ in R_q^2, computes $m_j = s_j u$, sample some uniform noise $E_j \leftarrow\!\!\$\ R_q$ such that $\|E_j\|_\infty \leq 2^{\mathtt{sec}}(B_{\mathtt{Dec}}/p\xi)$ for statistical security parameter sec and noise-bound $B_{\mathtt{Dec}} = \max\|v - su\|_\infty$, then outputs $\mathtt{ds}_j = t_j = m_j + pE_j$.

- Comb, on input the ciphertext (u, v) and the set of decryption shares $\{\mathtt{ds}_j\}_{j \in [\xi]}$, outputs the message $m = (v - t \mod q) \mod p$, where $t = t_1 + t_2 + \cdots + t_\xi$.

The following theorem for the distributed decryption protocol follows from the works by Damgård *et al.* [DPSZ12, Theorem 4] and [DKL⁺13, Appendix G].

Theorem 2 (Correctness and Simulatability of Distributed BGV). *Let* sec *be the statistical security parameter. The distributed BGV encryption scheme is* correct *(Definition 7 in 2.5) if* $\|v - t\|_\infty \leq (1 + 2^{\mathtt{sec}})B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$*, and is* decryption simulatable *against passive adversaries (Definition 9 in 2.5).*

3.2 Lattice-Based Commitments

Let R_q be defined as above for a fixed N and let $\mathcal{N}_{\sigma_{\mathrm{C}}}$ be a Gaussian distribution over R_q with standard deviation σ_{C}. The commitment scheme follows Definition 10 and consists of three algorithms: key generation ($\mathtt{KeyGen}_{\mathrm{C}}$), committing ($\mathtt{Com}$) and opening ($\mathtt{Open}$), where:

$\mathtt{KeyGen}_{\mathrm{C}}$ outputs a public key $\mathtt{pk}$ which allows to commit to messages in R_q^ℓ using randomness in $S_{B_{\mathtt{Com}}}^k$. We define

$$\begin{aligned} \boldsymbol{A}_1 &= \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{A}_1' \end{bmatrix} && \text{where } \boldsymbol{A}_1' \leftarrow\!\$\ R_q^{n\times(k-n)} \\ \boldsymbol{A}_2 &= \begin{bmatrix} \boldsymbol{0}^{\ell\times n} & \boldsymbol{I}_\ell & \boldsymbol{A}_2' \end{bmatrix} && \text{where } \boldsymbol{A}_2' \leftarrow\!\$\ R_q^{l\times(k-n-\ell)}, \end{aligned}$$

for height $n+\ell$ and width k and let $\mathtt{pk}$ be $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \end{bmatrix}$.

$\mathtt{Com}$ commits to messages $\boldsymbol{m} \in R_q^\ell$ by sampling an $\boldsymbol{r}_{\boldsymbol{m}} \leftarrow\!\$\ S_{B_{\mathtt{Com}}}^k$ and computes

$$\mathtt{Com}_{\mathtt{pk}}(\boldsymbol{m}; \boldsymbol{r}_{\boldsymbol{m}}) = \boldsymbol{A} \cdot \boldsymbol{r}_{\boldsymbol{m}} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{m} \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} = [\![\boldsymbol{m}]\!].$$

$\mathtt{Com}$ outputs commitment $[\![\boldsymbol{m}]\!]$ and opening $\boldsymbol{d} = (\boldsymbol{m}, \boldsymbol{r}_{\boldsymbol{m}}, 1)$.

$\mathtt{Open}$ verifies whether an opening $(\boldsymbol{m}, \boldsymbol{r}_{\boldsymbol{m}}, f)$, with $f \in \bar{\mathcal{C}}$, is a valid opening of $[\![\boldsymbol{m}]\!]$ for the public key $\mathtt{pk}$ by checking that $\|\boldsymbol{r}_{\boldsymbol{m}}[i]\| \le 4\sigma_{\mathrm{C}}\sqrt{N}$, for $i \in [k]$, and if

$$f \cdot \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} \stackrel{?}{=} \boldsymbol{A} \cdot \boldsymbol{r}_{\boldsymbol{m}} + f \cdot \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{m} \end{bmatrix}.$$

$\mathtt{Open}$ outputs 1 if all these conditions holds, and 0 otherwise.

For any commitment generated with $\mathtt{Com}$, the algorithm $\mathtt{Open}$ will accept (except with negligible probability) by setting $f = 1$.

The following theorem for the security of the commitment scheme follows from Baum *et al.* [BDL+18, Lemma 6 and Lemma 7].

Theorem 3 (Hiding and Binding of the Commitment Scheme). *The commitment scheme is* hiding *(Definition 12 in 2.6) if the* $\mathsf{DKS}^\infty_{n+\ell,k,\beta_\infty}$ *problem is hard, and the scheme is* binding *(Definition 13 in 2.6) if the* $\mathsf{SKS}^2_{n,k,16\sigma_{\mathrm{C}}\sqrt{\nu N}}$ *problem is hard.*

The commitments [BDL+18] have a weak additively homomorphic property:

Proposition 1. *Let* $[\![\boldsymbol{m}]\!] = \mathtt{Com}(\boldsymbol{m}; \boldsymbol{r}_{\boldsymbol{m}})$ *be a commitment with opening* $(\boldsymbol{m}, \boldsymbol{r}_{\boldsymbol{m}}, f)$ *and let* $[\![\boldsymbol{m}']\!] = \mathtt{Com}(\boldsymbol{m}'; \boldsymbol{0})$. *Then* $[\![\boldsymbol{m}]\!] - [\![\boldsymbol{m}']\!]$ *has the opening* $(\boldsymbol{m} - \boldsymbol{m}', \boldsymbol{r}_{\boldsymbol{m}}, f)$.

The proof follows from the linearity of the verification algorithm.

Prover$(\{(\boldsymbol{m}_i, \boldsymbol{r}_i)\}_{i\in[\hat{n}]}; \{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in[\hat{n}]})$		Verifier$(\{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in\hat{n}})$
$\boldsymbol{y}_i \leftarrow\!\!\$\, \mathcal{N}^k_{\sigma_\mathrm{C}}, i \in [\hat{n}]$		
$\boldsymbol{t}_i = \boldsymbol{A}_1\boldsymbol{y}_i, i \in [\hat{n}]$		
$\boldsymbol{u} = \boldsymbol{A}_2((\sum_{i\neq\hat{n}} \alpha_i\boldsymbol{y}_i) - \boldsymbol{y}_{\hat{n}})$	$\xrightarrow{\{\boldsymbol{t}_i\}_{i\in[\hat{n}]}, \boldsymbol{u}}$	
	$\xleftarrow{\beta}$	$\beta \xleftarrow{\$} \mathcal{C}$
$\boldsymbol{z}_i = \boldsymbol{y}_i + \beta\boldsymbol{r}_i, i \in [\hat{n}]$		$\hat{\boldsymbol{u}} = \boldsymbol{u} + \beta((\sum_{i\neq\hat{n}} \alpha_i\boldsymbol{c}_{i,2}) - \boldsymbol{c}_{\hat{n},2})$
For all i in $[\hat{n}]$:		
Abort if $\mathrm{Rej}(\boldsymbol{z}_i, \beta\boldsymbol{r}_i, \sigma_\mathrm{C}) = 1$.		
	$\xrightarrow{\{\boldsymbol{z}_i\}_{i\in[\hat{n}]}}$	**return** Accept iff:
		1 : $\forall i, j : \|z_{i,j}\|_2 \overset{?}{\leq} B$
		2 : $\forall i : \boldsymbol{A}_1\boldsymbol{z}_i \overset{?}{=} \boldsymbol{t}_i + \beta\boldsymbol{c}_{i,1}$
		3 : $\hat{\boldsymbol{u}} \overset{?}{=} \boldsymbol{A}_2((\sum_{i\neq\hat{n}} \alpha_i\boldsymbol{z}_i) - \boldsymbol{z}_{\hat{n}})$

Fig. 2. Protocol Π_{LIN} is a Sigma-protocol to prove the relation R_{LIN}.

3.3 Zero-Knowledge Proof of Linear Relations

Assume that there are $\hat{n}$ commitments

$$[\![\boldsymbol{m}_i]\!] = \begin{bmatrix} \boldsymbol{c}_{i,1} \\ \boldsymbol{c}_{i,2} \end{bmatrix}, \text{ for } 1 \leq i \leq \hat{n} \text{ where } \boldsymbol{c}_{i,2} \in R_q^\ell.$$

For the public scalar vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{\hat{n}-1}) \in R_q^{\hat{n}-1}$ the prover wants to demonstrate that the following relation R_{LIN} holds:

$$R_{\mathrm{LIN}} = \left\{ (x, w) \;\middle|\; \begin{array}{c} x = (\mathtt{pk}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in[\hat{n}]}, \boldsymbol{\alpha}) \wedge w = (f, \{\boldsymbol{m}_i, \boldsymbol{r}_i\}_{i\in[\hat{n}]}) \;\wedge \\ \forall i \in [\hat{n}] : \mathtt{Open}_{\mathtt{pk}}([\![\boldsymbol{m}_i]\!], \boldsymbol{m}_i, \boldsymbol{r}_i, f) = 1 \wedge \boldsymbol{m}_{\hat{n}} = \sum_{i=1}^{\hat{n}-1} \alpha_i\boldsymbol{m}_i \end{array} \right\}.$$

Π_{LIN} in Figure 2 is a zero-knowledge proof of knowledge (ZKPoK) of this relation (it is a directly extended version of the proof of linearity in [BDL+18]). The relation R_{LIN} is relaxed because of the additional factor f in the opening, which appears in the soundness proof. It does not show up in the protocol Π_{LIN}, because an honest prover will implicitly use $f = 1$.

The bound is $B = 2\sigma_\mathrm{C}\sqrt{N}$ and the Π_{LIN}-protocol produces a proof transcript of the form $\pi_{\mathrm{LIN}} = ((\{\boldsymbol{t}_i\}_{i\in[\hat{n}]}, u), \beta, (\{\boldsymbol{z}_i\}_{i\in[\hat{n}]}))$.

The following theorem for the security of zero-knowledge proof of linear relations is a direct adaption of Baum *et al.* [BDL+18, Lemma 8].

Theorem 4 (Security of Zero-Knowledge Proof of Linear Relations). *The zero-knowledge proof of linear relations is* complete *(Definition 15 in 2.7) if the randomness $\boldsymbol{r}_i$ is bounded by $B_{\mathtt{Com}}$ in the ℓ_∞ norm, it is* special sound

(Definition 16 in 2.7) if the $\mathsf{SKS}^2_{n,k,4\sigma_{\mathrm{C}}\sqrt{N}}$ *problem is hard, and it is statistical* honest-verifier zero-knowledge *(Definition 17 in 2.7). The probability of success is* $(1/M)^{\hat{n}}$*, where the constant* M *is computed as in Equation 1.*

Even though the success probability is $(1/M)^{\hat{n}}$, we will only use this protocol for small values of $\hat{n}$ and choose parameters so that the total rejection probability is small (around 1/3), which ensures that the protocol is efficient in practice.

When applying the Fiat-Shamir transform [FS87], we let β be the output of a hash-function applied to the first message and x. Then, the proof transcript is reduced to $\pi_L = (\beta, \{\boldsymbol{z}_i\}_{i\in[\hat{n}]})$ where β is of 2λ bits and each $\boldsymbol{z}_i$ is of size $kN\log_2(6\sigma_{\mathrm{C}})$ bits. We can compress $\boldsymbol{z}_i$ to $\hat{n}(k-n)N\log_2(6\sigma_{\mathrm{C}})$ bits by checking an approximate equality instead, as described in [ABG+21, Section 3.2]. We denote by

$$\begin{aligned}\pi_L &\leftarrow \mathit{\Pi}_{\mathrm{Lin}}(\{(\boldsymbol{m}_i, \boldsymbol{r}_i)\}_{i\in[\hat{n}]}; (\{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in[\hat{n}]})), \text{ and}\\ 0 \vee 1 &\leftarrow \mathit{\Pi}_{\mathrm{LinV}}(\{\alpha_i\}_{i\in[\hat{n}-1]}, \{[\![\boldsymbol{m}_i]\!]\}_{i\in[\hat{n}]}; \pi_L),\end{aligned}$$

the run of the proof and verification protocols, respectively, where the verification protocol $\mathit{\Pi}_{\mathrm{LinV}}$ reconstructs the first message using β, performs the verification as in the last step in Figure 2 and then checks that β was computed correctly with respect to the statement and the first message.

3.4 Exact Amortized Zero-Knowledge Proof of Short Openings

It is well-known that polynomials in R_q can be represented as vectors in $\mathbb{Z}_q^N$ and multiplication by a polynomial $\hat{a}$ in R_q can be expressed as a matrix-vector product with a nega-cyclic matrix $\hat{\boldsymbol{A}}$ in $\mathbb{Z}_q^{N\times N}$. Let $\boldsymbol{A}$ be a $r \times v$ matrix over R_q, that is, a $rN \times vN$ matrix over $\mathbb{Z}_q$. We will now consider how to prove generically that $\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i$ for a bounded $\boldsymbol{s}_i$, which is the same as proving correct multiplication over the ring R_q of a public polynomial a and a secret and bounded polynomials s_i resulting in public polynomials t_i.

Bootle *et al.* [BLNS21] give an efficient amortized sublinear zero-knowledge protocol for proving the knowledge of short vectors $\boldsymbol{s}_i$ and $\boldsymbol{e}_i$ over $\mathbb{Z}_q$ satisfying $\boldsymbol{A}\boldsymbol{s}_i + \boldsymbol{e}_i = \boldsymbol{t}_i$. Here we adapt their techniques for the case where $\boldsymbol{e}_i$ is zero, and prove that $\|\boldsymbol{s}_i\|_\infty \leq 1$. We further modify their protocol for amortized proofs to benefit from smaller parameters due to the small bound on $\boldsymbol{s}_i$, while their amortized protocol was optimized for larger bounds[3].

We first explain the main idea of [BLNS21] for proving knowledge of one preimage $\boldsymbol{s}$ of $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s}$ and then how it generalizes to an amortized proof for τ elements with sublinear communication.

The approach follows an ideal linear commitments-technique with vector commitments $\mathtt{Com}_{\mathtt{L}}(\cdot)$ over $\mathbb{Z}_q$. The prover initially commits to the vector $\boldsymbol{s}$ as well as an auxiliary vector $\boldsymbol{s}_0$ of equal length. Implicitly, this defines a vector

[3] The authors of [BLNS21] mention that this optimization is possible, but neither present the modified protocol nor a proof.

```
1 :  P has input $\{\boldsymbol{s}_i\}_{i=1}^{\tau}; (\boldsymbol{A}, \{\boldsymbol{t}_i\}_{i=1}^{\tau})$ while V has input $\boldsymbol{A}, \{\boldsymbol{t}_i\}_{i=1}^{\tau}$
2 :  P samples $\boldsymbol{s}_0 \leftarrow\!\$\; \mathbb{Z}_q^{vN}$, $\boldsymbol{h} \leftarrow\!\$\; \mathbb{Z}_q^{2vN}$, $\boldsymbol{r}_0 \leftarrow\!\$\; \mathbb{Z}_q^{\eta}$, $\boldsymbol{r}_h \leftarrow\!\$\; \mathbb{Z}_q^{\eta}$ and
     $\boldsymbol{r}_{i,j} \leftarrow\!\$\; \mathbb{Z}_q^{\eta}$ for each $i \in [\tau], j \in \{0,1,2\}$.
3 :  P computes $\boldsymbol{f}(X) = \sum_{i=0}^{\tau} \boldsymbol{s}_i \ell_i(X)$ and $\boldsymbol{d} = -\boldsymbol{A}\boldsymbol{s}_0$ and defines $\boldsymbol{v}_{i,j}$ using
         $\frac{1}{\ell_0(X)} \cdot \prod_{h \in \{-1,0,1\}} \left[\boldsymbol{f}(X) - h \cdot \boldsymbol{1}\right] = \sum_{j=0}^{2} \sum_{i=1}^{\tau} \boldsymbol{v}_{i,j} \ell_i(X) \ell_0(X)^j$
4 :  P computes $\boldsymbol{H}_0 \leftarrow \texttt{Encode}(\boldsymbol{s}_0, \boldsymbol{0}, \boldsymbol{r}_0)$, $\boldsymbol{H} \leftarrow \texttt{Encode}(\boldsymbol{h}, \boldsymbol{r}_h)$ as well as
     $\boldsymbol{H}_{i,j} = \texttt{Encode}(\delta_j \cdot \boldsymbol{s}_i, \boldsymbol{v}_{i,j}, \boldsymbol{r}_{i,j})$ for each $i \in [\tau], j \in \{0,1,2\}$.
5 :  P computes $\boldsymbol{E} = \texttt{RowsToMatrix}(\boldsymbol{H}, \boldsymbol{H}_0, \boldsymbol{H}_{1,0}, \boldsymbol{H}_{1,1}, \ldots, \boldsymbol{H}_{\tau,2})$,
     $\mathcal{M} = \texttt{MerkleTree}(\texttt{CommitToColums}(\boldsymbol{E}))$ and $(c_{\boldsymbol{d}}, r_{\boldsymbol{d}}) \leftarrow \texttt{Com}_{\texttt{Aux}}(\boldsymbol{d})$.
6 :  P sends $\mathcal{M}, c_{\boldsymbol{d}}$ to V
7 :  V samples $x \leftarrow\!\$\; \mathbb{Z}_q^* \setminus \{a_1, \ldots, a_\tau\}$ and $\beta_0, \beta_{1,0}, \ldots, \beta_{\tau,2} \leftarrow\!\$\; \mathbb{Z}_q^*$ and
       sends $x, \beta_0, \beta_{1,0}, \ldots, \beta_{\tau,2}$ to P
8 :  P computes $\bar{\boldsymbol{f}} = \boldsymbol{f}(x)$, $\bar{\boldsymbol{r}}_f = \ell_0(x)\boldsymbol{r}_0 + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x) \ell_0(x)^j \boldsymbol{r}_{i,j}$,
     $\bar{\boldsymbol{h}} = \boldsymbol{h} + \beta_0 \boldsymbol{s}_0 + \sum_{j=0}^{2} \sum_{i=1}^{\tau} \beta_{i,j}(\delta_j \cdot \boldsymbol{s}_i, \boldsymbol{v}_{i,j})$, $\bar{\boldsymbol{r}}_h = \boldsymbol{r}_h + \beta_0 \boldsymbol{r}_0 + \sum_{j=0}^{2} \sum_{i=1}^{\tau} \beta_{i,j} \boldsymbol{r}_{i,j}$
9 :  P sends $\bar{\boldsymbol{f}}, \bar{\boldsymbol{r}}_f, \bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h, r_{\boldsymbol{d}}$ to V
10 : V samples $I \leftarrow\!\$\; [l]^{\eta}$, $|I| = \eta$ and sends $I$ to P
11 : P computes and sends $\boldsymbol{E}|_I, \texttt{MerklePaths}_I$ to V
12 : V runs $\texttt{Verify}(\boldsymbol{E}|_I, \mathcal{M}, \texttt{MerklePaths}_I)$ and checks that
     $\texttt{Open}_{\texttt{Aux}}\left(c_{\boldsymbol{d}}, \frac{1}{\ell_0(x)} \cdot \left(\sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(x) - \boldsymbol{A}\bar{\boldsymbol{f}}\right), r_{\boldsymbol{d}}\right) = 1$,
     $\texttt{Encode}\left(\bar{\boldsymbol{f}}, \frac{1}{\ell_0(x)} \cdot \prod_{h \in \{-1,0,1\}} \left[\bar{\boldsymbol{f}} - h \cdot \boldsymbol{1}\right], \bar{\boldsymbol{r}}_f\right)\Big|_I$
         $\stackrel{?}{=} \ell_0(x) \cdot \boldsymbol{H}_0\big|_I + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x) \ell_0(x)^j \boldsymbol{H}_{i,j}|_I$
     $\texttt{Encode}\left(\bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h\right)\Big|_I \stackrel{?}{=} \boldsymbol{H} + \beta_0 \boldsymbol{H}_0\big|_I + \sum_{i=1}^{\tau} \sum_{j=0}^{2} \beta_{i,j} \boldsymbol{H}_{i,j}\big|_I$
```

Fig. 3. The protocol Π_{AEx} is an exact amortized zero-knowledge proof of knowledge of ternary openings. δ_x is 1 if $x = 0$ and 0 otherwise. $(\texttt{Com}_{\texttt{Aux}}, \texttt{Open}_{\texttt{Aux}})$ is an arbitrary commitment scheme.

of polynomials $\boldsymbol{f}(X) = \boldsymbol{s}_0(X) + \boldsymbol{s}$ for the prover. Now consider the vector of polynomials $\boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \mathbf{1}) \circ (\boldsymbol{f}(X) + \mathbf{1})$, where $\circ$ denote the coordinate-wise product, then we can see that the coefficients of X^0 are exactly $\boldsymbol{s} \circ (\boldsymbol{s} - \mathbf{1}) \circ (\boldsymbol{s} + \mathbf{1})$ and therefore $\mathbf{0}$ if and only if the aforementioned bound on $\boldsymbol{s}$ holds. In that case, each aforementioned polynomial in $\boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \mathbf{1}) \circ (\boldsymbol{f}(X) + \mathbf{1})$ is divisible by X. Therefore, the prover computes the coefficient vectors $\boldsymbol{v}_2, \boldsymbol{v}_1, \boldsymbol{v}_0$ of

$$1/X \cdot \boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \mathbf{1}) \circ (\boldsymbol{f}(X) + \mathbf{1}) = \boldsymbol{v}_2 X^2 + \boldsymbol{v}_1 X + \boldsymbol{v}_0$$

and commits to these. Additionally, define the value $\boldsymbol{d} = \boldsymbol{t} - \boldsymbol{A}\boldsymbol{f} = -\boldsymbol{A}\boldsymbol{s}_0$, which the prover also commits to.

The verifier now sends a challenge x, for which the prover responds with $\overline{\boldsymbol{f}} = \boldsymbol{f}(x)$. Additionally, the prover uses the linear property of the commitment scheme to show that:

1. $\texttt{Com}_{\texttt{L}}(\boldsymbol{s}_0) \cdot x + \texttt{Com}_{\texttt{L}}(\boldsymbol{s})$ opens to $\overline{\boldsymbol{f}}$.
2. $\texttt{Com}_{\texttt{L}}(\boldsymbol{v}_2) \cdot x^2 + \texttt{Com}_{\texttt{L}}(\boldsymbol{v}_1) \cdot x + \texttt{Com}_{\texttt{L}}(\boldsymbol{v}_0)$ opens to $\frac{1}{x} \cdot \overline{\boldsymbol{f}} \circ (\overline{\boldsymbol{f}} + 1) \circ (\overline{\boldsymbol{f}} - 1)$.

The prover additionally opens the commitment to $\boldsymbol{d}$ and the verifier checks that it opens to $\frac{1}{x} \cdot (\boldsymbol{t} - \boldsymbol{A}\overline{\boldsymbol{f}})$. Here, the first two commitment openings allow us to deduce that the correct $\overline{f}$ is sent by the prover and that the values committed as $\boldsymbol{s}$ are indeed commitments to $\{-1, 0, 1\}$. Then, from opening $\boldsymbol{d}$ we get that the committed $\boldsymbol{s}$ is indeed the preimage of $\boldsymbol{t}$ under $\boldsymbol{A}$.

The ideal linear commitments in [BLNS21] get realized using an Encode-then-Hash commitment scheme. In this commitment scheme, the prover commits to vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{Z}_q^g$ as follows:

1. Sample n random vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n \in \mathbb{Z}_q^\eta$
2. Let $\texttt{Encode}$ be the encoding function of an $[l, g + \eta, d]$ Reed-Solomon Code with code-length l, message length $g + \eta$ and minimal distance d. Compute $\boldsymbol{e}_i \leftarrow \texttt{Encode}(\boldsymbol{x}_i \| \boldsymbol{r}_i)$ for each $i \in [n]$.
3. Construct the matrix $\boldsymbol{E} = \texttt{RowsToMatrix}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$ where $\boldsymbol{e}_i$ is row i.
4. Commit to each *column* of $\boldsymbol{E}$ using a cryptographic hash function, then compress all m commitments into a Merkle tree root $\mathcal{M}$.
5. Send $\mathcal{M}$ to the verifier.

For the prover to show to the verifier that $\boldsymbol{x}$ is an opening of the linear combination $\sum_{i=1}^{n} \gamma_i \boldsymbol{x}_i$:

1. The prover computes $\boldsymbol{r} = \sum_{i=1}^{n} \gamma_i \boldsymbol{r}_i$ and sends $\boldsymbol{r}$ to the verifier.
2. The verifier chooses a subset I of size η from $[l]$.
3. The prover opens the commitment for each column $i \in I$ of $\boldsymbol{E}$ and proves that it lies in the Merkle tree $\mathcal{M}$ by revealing the path.
4. The verifier checks that $\texttt{Encode}(\boldsymbol{x} \| \boldsymbol{r})$ coincides at position i with the respective linear combination of all n opened values in column i of $\boldsymbol{E}$.

This is a proof of the respective statement due to the random choice of the set I. Intuitively, if each row of $\boldsymbol{E}$ is in the code[4], but they do not sum up to $\boldsymbol{x}$, then the linear combination of the codewords in $\boldsymbol{E}$ must differ from $\texttt{Encode}(\boldsymbol{x}\|\boldsymbol{r})$ in at least d positions, which is the minimum distance of the code. By the random choice of I and by setting η appropriately, the verifier would notice such a disagreeing entry with high probability. At the same time, because only η columns of $\boldsymbol{E}$ are opened, this leaks no information about the vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

For the case of more than one secret, the prover wants to show that $\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i$ for τ values $\boldsymbol{t}_i$ known to the verifier, subject to $\boldsymbol{s}_i$ again being ternary vectors. Here, the goal is to establish the latter for all $\boldsymbol{t}_i$ simultaneously while verifying only one equation and sending only one vector $\overline{\boldsymbol{f}}$. Towards this, the prover as before commits to $\boldsymbol{s}_i$ as well as an additional blinding value $\boldsymbol{s}_0$. Let $a_1, \ldots, a_\tau \in \mathbb{Z}_q$ be distinct interpolation points and define the ith Lagrange interpolation polynomial

$$\ell_i(X) = \prod_{i \neq j} \frac{X - a_j}{a_i - a_j}.$$

Additionally, let $\ell_0(X) = \prod_{i=1}^{\tau}(X - a_i)$. Then every $f \in \mathbb{Z}_q[X]/\ell_0(X)$ can be written uniquely as $f(X) = \sum_{i=1}^{\tau} \lambda_i \ell_i(X)$ and any $g \in \mathbb{Z}_q[X]/\ell_0(X)^b$ as a linear combination of $\{\ell_i(X)\ell_0(X)^j\}_{j=0}^{b-1}$. If we now, more generally, define the polynomial

$$\boldsymbol{f}(X) = \sum_{i=0}^{\tau} \boldsymbol{s}_i \ell_i(X),$$

then we observe that $\boldsymbol{f}(X) \circ (\boldsymbol{f}(X) - \mathbf{1}) \circ (\boldsymbol{f}(X) + \mathbf{1})$ is divisible by $\ell_0(X)$ iff all $\ell_i(X)$-coefficients of $\boldsymbol{f}(X)$ for $i \in [\tau]$ are 0. Additionally, since $\ell_i(X) \cdot \ell_j(X) = 0 \bmod \ell_0(X)$ if $i, j \in [n], i \neq j$ this then also implies that the $\boldsymbol{s}_i$ are ternary. Moreover, we only have to commit to additional $3 \cdot \tau$ coefficients of $\{\ell_i(X)\ell_0(X)^j\}_{j=0}^{b-1}$ to prove well-formedness of any evaluation of $\boldsymbol{f}(X)$ sent by the prover.

The protocol is described in detail in Figure 3. As our construction substantially deviates from that of [BLNS21] we show that the protocol indeed is a ZKPoK. Perfect completeness is straightforward, so we focus on soundness and special honest-verifier zero-knowledge.

Lemma 1 (Soundness in $\Pi_{\mathbf{AEx}}$). *Let* $\texttt{Encode}$ *be the encoding function of a Reed-Solomon code of dimension $k' = vN + \eta$ and length l. Furthermore, let $k' \leq k \leq l < q$. Suppose that there is an efficient deterministic prover* $\texttt{P}^*$ *convincing an honest verifier in the protocol in Figure 3 on input $\boldsymbol{A}, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_\tau$*

[4] For the proof to work, the verifier additionally has to verify this claim or rather, that all rows are close to actual codewords. One mechanism to achieve this is to commit to an additional auxiliary row and also open a random linear combination of all rows, including the auxiliary row. This will be more clear in the soundness proof of our protocol.

with probability

$$\epsilon > 2{\cdot}\max\left\{2\left(\frac{k}{l-\eta}\right)^{\eta}, \frac{1}{q-\tau} + \left(1-\frac{k-k'}{6l}\right)^{\eta}, 2\cdot\left(1-\frac{2(k-k')}{3l}\right)^{\eta}, \frac{18\tau}{q-\tau}\right\}.$$

Then there exists an efficient probabilistic extractor $\mathcal{E}$ which, given access to $\mathtt{P}^*$ either produces vectors $\boldsymbol{s}_i \in \{-1,0,1\}^{vN}$ such that $\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i$ for all $i \in [\tau]$, or breaks the binding property of the commitment scheme $(\mathtt{Com}_{\mathtt{Aux}}, \mathtt{Open}_{\mathtt{Aux}})$, or finds a hash collision in expected time at most $64T$ where

$$T := \frac{3}{\epsilon} + \frac{k-\eta}{\epsilon/2 - (k/(l-\eta))^{\eta}}$$

and running $\mathtt{P}^*$ takes unit time.

The proof for Lemma 1 as well as a proof of the zero-knowledge property are presented in Appendix C. The size of the amortized zero-knowledge proof in Figure 3 in terms of prover-to-verifier communication is

$$|c_{\boldsymbol{d}}| + |r_{\boldsymbol{d}}| + |\mathcal{M}| + (2vN + (3\tau+4)\eta)\log_2 q + \lambda\eta(1+\log_2 l) \text{ bits.} \qquad (2)$$

Theorem 5 (Security of Amortized Zero-Knowledge Proof of Exact Openings). *The amortized zero-knowledge proof of exact openings is* complete *(Definition 15 in 2.7) when the secrets $\boldsymbol{s}_i$ has ternary coefficients, it is* special sound *(Definition 16 in 2.7) if the $\mathsf{SKS}^2_{r,v,1}$ problem is hard (see Lemma 1), and it is statistically* honest-verifier zero-knowledge *(Definition 17 in 2.7).*

3.5 Amortized Zero-Knowledge Proof of Bounded Openings

Let $\boldsymbol{A}$ be a publicly known $r \times v$-matrix over R_q, let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_\tau$ be bounded elements in R_q^v and let $\boldsymbol{A}\boldsymbol{s}_i = \boldsymbol{t}_i$ for $i \in [\tau]$. Letting $\boldsymbol{S}$ be the matrix whose columns are $\boldsymbol{s}_i$ and $\boldsymbol{T}$ be the same matrix for $\boldsymbol{t}_i$, but defined over $\mathbb{Z}_q^N$ instead of R_q as in the previous subsection, then Baum *et al.* [BBC+18] give a efficient amortized zero-knowledge proof of knowledge for the relation

$$\mathcal{R}_{\text{ANEx}} = \left\{(x,w) \;\middle|\; \begin{array}{c} x = (\boldsymbol{A}, \boldsymbol{T}, \sigma_{\text{ANEx}}, B_{\text{ANEx}}) \wedge w = \boldsymbol{S} \;\wedge \\ \forall i \in [\tau]:\; \boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i \wedge ||s_{i,j}||_2 \le 2\cdot B_{\text{ANEx}} \end{array}\right\}.$$

The protocol $\mathit{\Pi}_{\text{ANEx}}$ is depicted in Figure 4. We can use a challenge matrix $\boldsymbol{C}$ with entries sampled from the set $\mathcal{C}_{\text{ANEx}} = \{0,1\}$, then this allows us to choose the parallel protocol instances $\hat{n} \ge \lambda + 2$ for security parameter λ. Denote by

$$\pi_{\text{ANEx}} \leftarrow \mathit{\Pi}_{\text{ANEx}}(\boldsymbol{S}; (\boldsymbol{A}, \boldsymbol{T}, \sigma_{\text{ANEx}})), \text{ and } 0 \vee 1 \leftarrow \mathit{\Pi}_{\text{ANExV}}((\boldsymbol{A}, \boldsymbol{T}, B_{\text{ANEx}}); \pi_{\text{ANEx}}),$$

the run of the proof and verification protocols, respectively, where the $\mathit{\Pi}_{\text{ANEx}}$-protocol, using Fiat-Shamir, produces a proof of the form $\pi_{\text{ANEx}} = (\boldsymbol{C}, \boldsymbol{Z})$, where $\boldsymbol{C}$ is the output of a hash-function, and the $\mathit{\Pi}_{\text{ANExV}}$-protocol consists of the two checks in the last step in Figure 4. $\mathcal{N}_{\sigma_{\text{ANEx}}}$ is a Gaussian distribution over $\mathbb{Z}$ with

$$\begin{array}{lcl}
\underline{\text{Prover}(\boldsymbol{S}; (\boldsymbol{A}, \boldsymbol{T}, \sigma_{\text{ANEx}}))} & & \underline{\text{Verifier}(\boldsymbol{A}, \boldsymbol{T}, B_{\text{ANEx}})} \\
\forall i,j : y_{i,j} \leftarrow\!\!\$\, \mathcal{N}_{\sigma_{\text{ANEx}}} & & \\
\boldsymbol{W} = \boldsymbol{AY} & \xrightarrow{\quad\boldsymbol{W}\quad} & \\
 & & \boldsymbol{C} \leftarrow\!\!\$\, \mathcal{C}_{\text{ANEx}}^{\tau N \times \hat{n} N} \\
 & \xleftarrow{\quad\boldsymbol{C}\quad} & \\
\boldsymbol{Z} \leftarrow \boldsymbol{Y} + \boldsymbol{SC} & & \\
\text{Abort if Rej}(\boldsymbol{Z}, \boldsymbol{SC}, \sigma_{\text{ANEx}}) = 1. & & \\
 & \xrightarrow{\quad\boldsymbol{Z}\quad} & 1: \quad \forall i,j : \|z_{i,j}\|_2 \overset{?}{\leq} B_{\text{ANEx}} \\
 & & 2: \quad \boldsymbol{AZ} \overset{?}{=} \boldsymbol{TC} + \boldsymbol{W}
\end{array}$$

Fig. 4. Protocol Π_{ANEx} is the approximate amortized zero-knowledge proof of knowledge of bounded preimages for matrices and vectors over $\mathbb{Z}_q$.

standard deviation σ_{ANEx}, and the verification bound is $B_{\text{ANEx}} = \sqrt{2N}\sigma_{\text{ANEx}}$. Note that σ_{ANEx}, and hence B_{ANEx}, depends on the norm of $\boldsymbol{S}$ (see Section 2.2). This means that the bound we can prove for each $\|s_{i,j}\|_2$ depends on the number of equations τ in the statement to be proved.

The following theorem for the security of the amortized zero-knowledge proof of bounded openings follows from Baum *et al.* [BBC$^+$18, Lemma 3].

Theorem 6 (Security of Amortized Zero-Knowledge Proof of Bounded Openings). *The amortized zero-knowledge proof of bounded openings is* complete *(Definition 15 in 2.7) if the secrets in* $\boldsymbol{S}$ *are bounded by* $B_{\texttt{Com}}$ *in the* ℓ_∞ *norm, it is* special sound *(Definition 16 in 2.7) if the* $\mathsf{SKS}^2_{n,k,2kB_{\text{ANEx}}}$ *problem is hard, and is statistical* honest-verifier zero-knowledge *(Definition 17 in 2.7). The probability of success is* $1/M$*, as computed in Equation 1.*

Finally, we note that this protocol can be generalized to check for different norms (and using different standard deviation) in each row or column of $\boldsymbol{Y}$ and $\boldsymbol{Z}$ depending on varying norms of the secrets $s_{i,j}$, see [BEPU$^+$20, Section 5] for details.

4 Verifiable Shuffle of BGV Ciphertexts

The recent work by Aranha *et al.* [ABG$^+$21] presents an efficient protocol Π_{SHUF} for a shuffle of openings of lattice-based commitments using zero-knowledge proofs of linear relations.

More concretely, the authors present a proof for the following relation R_{Shuf}:

$$R_{\text{Shuf}} = \left\{ (x,w) \left| \begin{array}{l} x = (\llbracket m_1 \rrbracket, \dots, \llbracket m_\tau \rrbracket, \hat{m}_1, \dots, \hat{m}_\tau), \\ w = (\pi, f_1, \dots, f_\tau, \boldsymbol{r}_1, \dots, \boldsymbol{r}_\tau), \pi \in S_\tau, \hat{m}_i \in R_q \\ \forall i \in [\tau]: \ f_i \cdot \llbracket m_{\pi^{-1}(i)} \rrbracket = f_i \cdot \begin{bmatrix} \boldsymbol{c}_{1,\pi^{-1}(i)} \\ c_{2,\pi^{-1}(i)} \end{bmatrix} = \boldsymbol{A}\boldsymbol{r}_i + f_i \cdot \begin{bmatrix} \boldsymbol{0} \\ \hat{m}_i \end{bmatrix} \\ \wedge \ ||\boldsymbol{r}_i[j]|| \leq 4\sigma_C \sqrt{N} \end{array} \right. \right\}.$$

In their work, they show the following result:

Theorem 7 (Security of $\varPi_{\text{Shuf}}$). *Assume that* $(\texttt{KeyGen}_\text{C}, \texttt{Com}, \texttt{Open})$ *is a secure commitment scheme with $\varPi_{\text{Lin}}$ as a HVZK Proof of Knowledge of linear relation with soundness error ϵ. Then there exists a protocol $\varPi_{\text{Shuf}}$ that is an HVZK PoK for the relation $\mathcal{R}_{\text{Shuf}}$ with soundness error $(\tau^\delta + 1)/|R_q| + 4\tau\epsilon$.*

We now extend their protocol, allowing to verifiably shuffle elements that are vectors in R_q^ℓ instead of the original elements from R_q.

4.1 The Extended Shuffle Protocol

We are now in a situation where both prover and verifier are given a list of commitments $\llbracket \boldsymbol{m}_1 \rrbracket, \dots, \llbracket \boldsymbol{m}_\tau \rrbracket$ as well as potential messages $(\hat{\boldsymbol{m}}_1, \dots, \hat{\boldsymbol{m}}_\tau)$ from R_q^ℓ. The prover additionally obtains openings $\boldsymbol{m}_i, \boldsymbol{r}_i, f_i$ and wants to prove that the set of plaintext elements are the same set as the underlying elements of the commitments for some secret permutation π of the indices in the lists. The protocol of Aranha *et al.* does not work in this setting, as their technique crucially requires that the plaintexts are from R_q. Thus, our goal is to prove

$$R^\ell_{\text{Shuf}} = \left\{ (x,w) \left| \begin{array}{l} x = (\llbracket \boldsymbol{m}_1 \rrbracket, \dots, \llbracket \boldsymbol{m}_\tau \rrbracket, \hat{\boldsymbol{m}}_1, \dots, \hat{\boldsymbol{m}}_\tau), \\ w = (\pi, f_1, \dots, f_\tau, \boldsymbol{r}_1, \dots, \boldsymbol{r}_\tau), \pi \in S_\tau, \hat{\boldsymbol{m}}_i \in R_q^\ell \\ \forall i \in [\tau]: \ f_i \cdot \llbracket \boldsymbol{m}_{\pi^{-1}(i)} \rrbracket = f_i \cdot \begin{bmatrix} \boldsymbol{c}_{1,\pi^{-1}(i)} \\ \boldsymbol{c}_{2,\pi^{-1}(i)} \end{bmatrix} = \boldsymbol{A}\boldsymbol{r}_i + f_i \cdot \begin{bmatrix} \boldsymbol{0} \\ \hat{\boldsymbol{m}}_i \end{bmatrix} \\ \wedge \ ||\boldsymbol{r}_i[j]|| \leq 4\sigma_C \sqrt{N} \end{array} \right. \right\}.$$

Towards proving this relation, we observe that instead of proving a shuffle on the vectors directly, it is sufficient to let the verifier choose a random element $h \leftarrow\!\!\$\ R_q$. Then instead of proving a shuffle on $\boldsymbol{m}_1, \dots, \boldsymbol{m}_\tau$, the prover instead performs the same proof on $\langle \boldsymbol{m}_1, \rho \rangle, \dots, \langle \boldsymbol{m}_\tau, \rho \rangle$ where $\rho = (1, h, \dots, h^{\ell-1})^\top$. The problem with this approach is that we must also be able to apply ρ to the commitments $\llbracket \boldsymbol{m}_1 \rrbracket, \dots, \llbracket \boldsymbol{m}_\tau \rrbracket$, without re-committing to the inner product and proving correctness in zero-knowledge.

Since each commitment $\llbracket \boldsymbol{m} \rrbracket$ can be written as

$$\begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} = \boldsymbol{A}\boldsymbol{r} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{m} \end{bmatrix},$$

we can write $\boldsymbol{c}_1 = \boldsymbol{A}_1\boldsymbol{r}$ and $\boldsymbol{c}_2 = \boldsymbol{A_2}\boldsymbol{r} + \boldsymbol{m}$. From this we can create a new commitment $\llbracket \langle \boldsymbol{\rho}, \boldsymbol{m} \rangle \rrbracket$ under the new commitment key $pk' = (\boldsymbol{A}_1, \rho\boldsymbol{A}_2)$ where

$\boldsymbol{c}_1' = \boldsymbol{c}_1$ remains the same, while we set $c_2' = \langle \rho, \boldsymbol{c}_2 \rangle$. Note that this does not increase the bound of the randomness of the commitment. Since

$$\boldsymbol{A}_2 = \begin{bmatrix} \boldsymbol{0}^{\ell \times n} & \boldsymbol{I}_\ell & \boldsymbol{A}_2' \end{bmatrix} \text{ where } \boldsymbol{A}_2' \in R_q^{l \times (k-n-\ell)},$$

it holds that

$$\boldsymbol{a}_2' = \boldsymbol{\rho} \boldsymbol{A}_2 = \begin{bmatrix} \boldsymbol{0}^n & \boldsymbol{\rho}^\top & \boldsymbol{\rho} \boldsymbol{A}_2' \end{bmatrix}.$$

It is easy to see that breaking the binding property for *pk′* is no easier than breaking the binding property for *pk*.

Proposition 2. *If there exists an efficient attacker $\mathcal{A}$ that breaks the binding property on commitments under the key pk′ with probability ϵ, then there exists an efficient algorithm $\mathcal{A}'$ that breaks the binding property on pk with the same probability.*

Proof. Assume that $\mathcal{A}$ outputs two valid $(\boldsymbol{r}_1, m_1, f_1), (\boldsymbol{r}_2, m_2, f_2)$ for *pk′*. This in particular implies that $f_1 c_2' = \langle \boldsymbol{a}_2', \boldsymbol{r}_1 \rangle + f_1 m_1$ and $f_2 c_2' = \langle \boldsymbol{a}_2', \boldsymbol{r}_2 \rangle + f_2 m_2$. Multiplying the first term with f_2 and the second with f_1 yields that

$$0 = \langle \boldsymbol{a}_2', (f_2 \boldsymbol{r}_1 - f_1 \boldsymbol{r}_2) \rangle + f_1 f_2 (m_1 - m_2).$$

Since $m_1 \neq m_2$ we have that $\langle \boldsymbol{a}_2', f_2 \boldsymbol{r}_1 - f_1 \boldsymbol{r}_2 \rangle \neq 0$.

Let $\boldsymbol{m}_1 = (f_1 c_2 - \boldsymbol{A}_2 r_1)/f_1$ and $\boldsymbol{m}_2 = (f_2 c_2 - \boldsymbol{A}_2 r_2)/f_2$. It is clear that $(\boldsymbol{r}_1, \boldsymbol{m}_1, f_1)$ and $(\boldsymbol{r}_2, \boldsymbol{m}_2, f_2)$ are valid openings for *pk*. We need to show that indeed $\boldsymbol{m}_1 \neq \boldsymbol{m}_2$.

Towards this, assume that $\boldsymbol{m}_1 = \boldsymbol{m}_2$. By multiplying the definition with $f_1 f_2$ we get that $\boldsymbol{0} = \boldsymbol{A}_2(f_1 \boldsymbol{r}_2 - f_2 \boldsymbol{r}_1)$. This implies that

$$\langle \boldsymbol{\rho}, \boldsymbol{A}_2(f_1 \boldsymbol{r}_2 - f_2 \boldsymbol{r}_1) \rangle = \langle \boldsymbol{\rho} \boldsymbol{A}_2, f_1 \boldsymbol{r}_2 - f_2 \boldsymbol{r}_1 \rangle = \langle \boldsymbol{a}_2', f_1 \boldsymbol{r}_2 - f_2 \boldsymbol{r}_1 \rangle = 0$$

which is a contradiction. □

Shuffle protocol. Given the above, we can now construct the protocol Π^ℓ_{SHUF}:

1. Initially, `P` and `V` hold $\{[\![\boldsymbol{m}_i]\!], \hat{\boldsymbol{m}}_i\}_{i \in [\tau]}$ for a public key $pk = (\boldsymbol{A}_1, \boldsymbol{A}_2)$ while the prover additionally has $\{\boldsymbol{m}_i, \boldsymbol{r}_i\}_{i \in [\tau]}, \pi \in S_\tau$.
2. `V` chooses $h \leftarrow\$ R_q$ and sends it to `P`. Both parties compute $\boldsymbol{\rho} \leftarrow (1, h, \ldots, h^{\ell-1})^\top$.
3. `P` and `V` for each $[\![\boldsymbol{m}_i]\!] = (\boldsymbol{c}_{1,i}, \boldsymbol{c}_{2,i})$ compute $[\![\langle \boldsymbol{\rho}, \boldsymbol{m}_i \rangle]\!] = (\boldsymbol{c}_{1,i}, \langle \boldsymbol{\rho}, \boldsymbol{c}_{2,i} \rangle)$.
4. `P`, `V` now run Π_{SHUF} on input commitments $\{[\![\langle \boldsymbol{\rho}, \boldsymbol{m}_i \rangle]\!]\}_{i \in [\tau]}$ and messages $\langle \boldsymbol{\rho}, \boldsymbol{m}_i \rangle$. `P` uses the same permutation π, randomness $\boldsymbol{r}_i$ as before. The commitment key $pk' = (\boldsymbol{A}_1, \boldsymbol{\rho} \boldsymbol{A}_2)$ is used by both parties.
5. If the protocol Π_{SHUF} accepts then `V` accepts, otherwise he rejects.

We now show the following:

Lemma 2 (Soundness in Π^ℓ_{Shuf}). *Assume that Π_{SHUF} that is an HVZK Proof of Knowledge for the relation $\mathcal{R}_{\text{SHUF}}$ with soundness error ϵ'. Then Π^ℓ_{SHUF} is a HVZK PoK for the relation R^ℓ_{SHUF} with soundness error $\epsilon = 2\epsilon' + 3\left(\frac{\ell-1}{q}\right)^N$.*

Proof. Completeness and Zero-Knowledge of Π^{ℓ}_{SHUF} follow immediately from the same properties of Π_{SHUF}. Thus, we focus now on knowledge soundness.

Let $\mathtt{P}^*$ be a prover that convinces a verifier on input x with probability $\nu > \epsilon$. For the proof, we will use the standard definition of proof of knowledge where there must exist an extractor $\mathcal{E}$ that succeeds with black-box access to $\mathtt{P}^*$ running in expected time $p(|x|)/(\nu - \epsilon)$ where p is a polynomial.

Towards constructing a simulator $\mathcal{E}$ we know that there exists an extractor $\mathcal{E}'$ for Π_{SHUF}. We construct $\mathcal{E}$ as the following loop, which restarts whenever the loop "aborts":

1. Run random protocol instances with $\mathtt{P}^*$ until a valid protocol instance with challenge h was generated. Do this at most $2/\epsilon$ steps, otherwise abort.
2. Run $\mathcal{E}'$ with the fixed h with $\mathtt{P}^*$ until it outputs $\pi, \{f_i, \boldsymbol{r}_i\}_{i\in[\tau]}$. If $\mathcal{E}'$ aborts, then abort. In parallel, start a new loop instance until $\mathcal{E}'$ finishes.
3. Let $\tilde{\boldsymbol{m}}_i = (f_i \boldsymbol{c}_{2,i} - \boldsymbol{A}_2 \boldsymbol{r}_i)/f_i$. If $\tilde{\boldsymbol{m}}_{\pi^{-1}(i)} = \hat{\boldsymbol{m}}_i$ for all $i \in [\tau]$ then output $\pi, \{f_i, \boldsymbol{r}_i\}_{i\in[\tau]}$, otherwise abort.

First, by the definition we observe that $\tilde{\boldsymbol{m}}_i = (f_i \boldsymbol{c}_{2,i} - \boldsymbol{A}_2 \boldsymbol{r}_i)/f_i$ is well-defined because f_i is invertible. If $\mathcal{E}$ outputs a value, then the output of $\mathcal{E}$ is a witness for the relation R^{ℓ}_{SHUF}. We now show a bound on the expected time per loop-instance, and that each loop with constant probability outputs a valid witness.

In the first step, we expect to find an accepting transcript after $1/\epsilon$ steps. Since we run this step for $2/\epsilon$ iterations, we will have found an accepting transcript with probability at least $1/2$ by Markov's inequality. Consider the matrix $\boldsymbol{H}$ where the rows are indexed by all choices h and the columns by the choices of the used shuffle proof. Then, by the heavy-row lemma [Dam10], with probability $\geq 1/2$ we will have chosen a value h such that the row of $\boldsymbol{H}$ contains $\epsilon/2 > \epsilon'$ 1s. In that case, $\mathcal{E}'$ will by definition output a valid witness in an expected number of $p(|x|)/(\nu - \epsilon') < p(|x|)/(\nu - \epsilon)$ steps, which is within the runtime budget. For the case it gets stuck, we start another loop which we run in parallel. Once $\mathcal{E}'$ has found an opening, then the computation in Step 3 is inexpensive. We now have to compute the abort probability of this step.

First, assume that $\mathcal{E}'$ outputs the same opening with probability at least $1/2$ in 2/3rds of the heavy rows. It can easily be shown that we can otherwise construct an algorithm that breaks the binding property of the commitment scheme with an expected constant number of calls to $\mathcal{E}'$ and by using Proposition 2. Moreover, by a counting argument, there must be $> \frac{3}{2}\left(\frac{\ell-1}{q}\right)^N$ heavy rows: Assume to the contrary that there are at most $\frac{3}{2}\left(\frac{\ell-1}{q}\right)^N$ heavy rows. Let each of the heavy rows have only ones (verifier always accepts), and each other row be filled with $\epsilon/2$ ones. This is the maximal case of having only $\frac{3}{2}\left(\frac{\ell-1}{q}\right)^N$ heavy rows. But then the acceptance probability can be at most

$$\frac{3}{2}\left(\frac{\ell-1}{q}\right)^N + \left[1 - \frac{3}{2}\left(\frac{\ell-1}{q}\right)^N\right] \cdot \epsilon/2 < \frac{3}{2}\left(\frac{\ell-1}{q}\right)^N + \epsilon/2 < \epsilon.$$

Assume that $\mathcal{E}'$ extracts a valid witness $\pi, \{f_i, \boldsymbol{r}_i\}_{i\in[\tau]}$ for input commitments $\{[\![\langle \boldsymbol{\rho}, \boldsymbol{m}_i \rangle]\!]\}_{i\in[\tau]}$ and messages $\langle \boldsymbol{\rho}, \hat{\boldsymbol{m}}_i \rangle$ while the extracted $\tilde{\boldsymbol{m}}_i = (f_i \boldsymbol{c}_{2,i} - \boldsymbol{A}_2 \boldsymbol{r}_i)/f_i$ do not form a permutation on the $\hat{\boldsymbol{m}}_i$. Then there exists an $i \in [\tau]$ such that

$$f_i \cdot \langle \rho, \boldsymbol{c}_{2,\pi^{-1}(i)} \rangle = \langle \rho \boldsymbol{A}_2, \boldsymbol{r}_i \rangle + f_i \langle \boldsymbol{\rho}, \hat{\boldsymbol{m}}_i \rangle$$

but

$$f_i \cdot \boldsymbol{c}_{2,\pi^{-1}(i)} = \boldsymbol{A}_2 \boldsymbol{r}_i + f_i(\hat{\boldsymbol{m}}_i + \boldsymbol{\delta})$$

where $\tilde{\boldsymbol{m}}_i = \hat{\boldsymbol{m}}_i + \boldsymbol{\delta}$ for a non-zero vector $\boldsymbol{\delta}$. Combining both equations, we get that $\boldsymbol{0} = \langle \boldsymbol{\rho}, \boldsymbol{\delta} \rangle$. This implies that the polynomial $\sum_{i=0}^{\ell-1} \boldsymbol{\delta}[i] X^i$ that has coefficients from $\boldsymbol{\delta}$ must be zero at point h whose powers generate the vector $\boldsymbol{\rho}$. Since this polynomial is of degree $\ell - 1$, by [ABG$^+$21, Lemma 2] it can be 0 in at most $(\ell-1)^N$ positions without being the 0-polynomial itself. But since the transcript is extractable and thus accepting for strictly more than $(\ell-1)^N$ choices of h (by our choice of the "default witness" from above), we must have that $\boldsymbol{\delta}$ was $\boldsymbol{0}$ to begin with. Therefore, Step 3 only aborts if we stumble upon a witness $\pi, \{f_i, \boldsymbol{r}_i\}_{i\in[\tau]}$ for R_{SHUF} that is not the "default witness" which occurs with constant probability only. □

Notation and communication. We denote by

$$\begin{aligned} &\pi_{\text{SHUF}} \leftarrow \Pi^{\ell}_{\text{SHUF}}((\{(\boldsymbol{m}_i, \boldsymbol{r}_i)\}_{i=1}^{\tau}, h); (\{[\![\boldsymbol{m}_i]\!]\}_{i=1}^{\tau}, \{\hat{\boldsymbol{m}}_i\}_{i=1}^{\tau})), \text{ and} \\ &0 \vee 1 \leftarrow \Pi^{\ell}_{\text{SHUFV}}((\{[\![\boldsymbol{m}_i]\!]\}_{i=1}^{\tau}, \{\hat{\boldsymbol{m}}_i\}_{i=1}^{\tau}); \pi_{\text{SHUF}}) \end{aligned}$$

the run of the proof and verification protocols of the shuffle, respectively.

We refer to Section 7 for sizes, parameters and a concrete instantiation of the shuffle protocol, as parameters depend on the full mix-net protocol and the decryption protocol, such as the number of servers involved.

4.2 Verifiable Shuffle of BGV Ciphertexts

The following mixing protocol is for the relation R_{MIX}:

$$R_{\text{MIX}} = \left\{ (x, w) \left| \begin{array}{l} x = (\boldsymbol{A}'', \boldsymbol{c}_1, \ldots, \boldsymbol{c}_\tau, \hat{\boldsymbol{c}}_1, \ldots, \hat{\boldsymbol{c}}_\tau, [\![\boldsymbol{c}'_1]\!], \ldots, [\![\boldsymbol{c}'_\tau]\!]), \\ w = (\pi, \boldsymbol{r}'_1, \ldots, \boldsymbol{r}'_\tau,), \pi \in S_\tau, \\ \forall i \in [\tau] : \boldsymbol{c}_i = \mathsf{Enc}(\mathsf{pk}, m_i), \boldsymbol{c}'_i = \mathsf{Enc}(\mathsf{pk}, 0), \\ [\![\boldsymbol{c}'_i]\!] = \boldsymbol{A}'' \boldsymbol{r}'_i, \|\boldsymbol{r}'_i\|_\infty \le B_\infty, \hat{\boldsymbol{c}}_{\pi(i)} = \boldsymbol{c}_i + \boldsymbol{c}'_i \end{array} \right. \right\}.$$

If the noise-level in all $\boldsymbol{c}_i$ and $\boldsymbol{c}'_i$ are bounded by $B_{\mathtt{Dec}}$, and $2B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$, then all $\boldsymbol{c}_i$ and $\hat{\boldsymbol{c}}_{\pi(i)}$ will, for some permutation π, decrypt to the same message m_i.

Public parameters. Let $p \ll q$ be primes, let R_q and R_p be defined as above for a fixed N, let $\mathtt{D}$ be a bounded distribution over R_q, and let $B_\infty \in \mathbb{N}$ be a bound. We assume properly generated keys and ciphertexts according to the `KeyGen` and `Enc` algorithms in Section 3.1.

Let $\mathcal{S}$ be the shuffle algorithm. Then $\mathcal{S}$ takes as input a set of τ publicly known BGV ciphertexts $\{\boldsymbol{c}_i\}_{i=1}^{\tau}$, where each ciphertext is of the form

$$\boldsymbol{c}_i = (u_i, v_i) = (ar_i + pe_{i,1}, br_i + pe_{i,2} + m_i),$$

where m_i in R_p is the encrypted message, $r_i \leftarrow\!\!\$\, R_q$ is a short element such that $\|r_i\|_\infty \le B_\infty$, and $e_{i,1}, e_{i,2} \leftarrow \mathtt{D}$ is some bounded random noise ensuring that the total noise in the ciphertext is bounded by $B_{\mathtt{Dec}}$.

Randomizing. First, $\mathcal{S}$ randomizes all the received ciphertexts and creates a new set of ciphertexts $\{\boldsymbol{c}'_i\}_{i=1}^{\tau}$ of the form

$$\boldsymbol{c}'_i = (u'_i, v'_i) = (ar'_i + pe'_{i,1}, br'_i + pe'_{i,2}),$$

where $r'_i, e'_{i,1}, e'_{i,2}$ are chosen as above. This corresponds to creating fresh, independent encryptions of 0. Observe that $\mathcal{S}$ will *not* publish these $\boldsymbol{c}'_i$.

Committing. $\mathcal{S}$ now commits to the $\boldsymbol{c}'_i$. Towards this, we re-write the commitment matrix from Section 3.2 for $\ell = 2$ and add the public key of the encryption scheme to get a $(n+2)\times(k+3)$ commitment matrix $\boldsymbol{A}''$, where $\boldsymbol{0}^n$ are row-vectors of length n, $\boldsymbol{a}_{1,1}, \boldsymbol{a}_{1,2}$ are column vectors of length n, $\boldsymbol{a}_{2,3}, \boldsymbol{a}_{3,3}$ are row vectors of length $k-n-2$ and $\boldsymbol{A}_{1,3}$ is of size $n \times (k-n-2)$. Then,

$$\begin{aligned}\mathtt{Com}(u'_i, v'_i) &= [\![(ar'_i + pe'_{i,1}, br'_i + pe'_{i,2})]\!] = \boldsymbol{A}''\boldsymbol{r}'_i \\ &= \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{a}_{1,1} & \boldsymbol{a}_{1,2} & \boldsymbol{A}_{1,3} & 0 & 0 & 0 \\ \boldsymbol{0}^n & 1 & 0 & \boldsymbol{a}_{2,3} & a & p & 0 \\ \boldsymbol{0}^n & 0 & 1 & \boldsymbol{a}_{3,3} & b & 0 & p \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_i \\ r'_i \\ e'_{i,1} \\ e'_{i,2} \end{bmatrix},\end{aligned}$$

where $\boldsymbol{r}_i \in R_q^k$ is the randomness used in the commitment. Further, let $[\![(u_i, v_i)]\!]_{\boldsymbol{0}}$ be the trivial commitment to (u_i, v_i) with no randomness. Then, given the commitment $[\![(u'_i, v'_i)]\!]$ and $[\![(u_i, v_i)]\!]_{\boldsymbol{0}}$ we can compute a commitment

$$[\![(\hat{u}_i, \hat{v}_i)]\!] = [\![(u_i, v_i)]\!]_{\boldsymbol{0}} + [\![(u'_i, v'_i)]\!].$$

Thus, the commitments $[\![(\hat{u}_i, \hat{v}_i)]\!]$ contain re-randomized encryptions of the original ciphertexts. $\mathcal{S}$ can therefore open a permutation of the $(\hat{u}_i, \hat{v}_i)$ and prove correctness of the shuffled opening using algorithm Π_{SHUF}^{ℓ}. To ensure correctness we have to additionally show that each u'_i, v'_i was created such that decryption is still correct.

Proving correctness of commitments. Let $\boldsymbol{A}''$ be the $(n+2)\times(k+3)$ matrix defined above. Then $\mathcal{S}$ needs to prove that, for all i, it knows secret short vectors $\boldsymbol{r}'_i$ of length $k+3$ that are solutions to the following equations:

$$\boldsymbol{t}_i = \boldsymbol{A}''\boldsymbol{r}'_i = [\![(ar'_i + pe'_{i,1}, br'_i + pe'_{i,2})]\!], \qquad \|\boldsymbol{r}'_i\|_\infty \le B_\infty.$$

To show this, $\mathcal{S}$ runs the Π_{AEx}-protocol for the inputs $\boldsymbol{A}''$, $\{\boldsymbol{r}'_i\}_{i=1}^{\tau}$, $\{\boldsymbol{t}_i\}_{i=1}^{\tau}$. Here, $\mathcal{S}$ uses the Fiat-Shamir transform to ensure non-interactivity of the proof.

We summarize the aforementioned in the following protocol Π_{Mix}:

1. $\mathcal{S}$ obtains as input the ciphertexts $\{\boldsymbol{c}_i\}_{i\in[\tau]} = \{(u_i, v_i)\}_{i\in[\tau]}$.
2. $\mathcal{S}$ for each $i \in [\tau]$ samples $r'_i, e'_{i,1}, e'_{i,2}$ as mentioned above. It then creates commitments $\{[\![u'_i, v'_i]\!] = [\![ar'_i + pe'_{i,1}, br'_i + pe'_{i,2}]\!]\}_{i\in[\tau]}$ using randomness $\boldsymbol{r}_i$ for each such commitment.
3. Let $\boldsymbol{t}_i = [\![u'_i, v'_i]\!]$ and $\boldsymbol{r}'_i = [\boldsymbol{r}_i^\top, r'_i, e_{i,1}, e_{i,2}]^\top$. Then $\mathcal{S}$ computes $\pi_{\text{AEx}} \leftarrow \Pi_{\text{AEx}}$ for matrix $\boldsymbol{A}''$, input vectors $\{\boldsymbol{r}'_i\}$, target vectors $\{\boldsymbol{t}_i\}$ and bound B_∞.
4. Let $\hat{\boldsymbol{c}}_i = (u_i + u'_i, v_i + v'_i)$ and L be a random permutation of $\{\hat{\boldsymbol{c}}_i\}_{i\in[\tau]}$. Then $\mathcal{S}$ computes $\pi_{\text{Shuf}} \leftarrow \Pi^\ell_{\text{Shuf}}$ with input commitments $\{[\![(\hat{u}_i, \hat{v}_i)]\!]\}_{i\in[\tau]}$, commitment messages $\{(u_i, v_i)\}_{i\in[\tau]}$, commitment randomness $\{\boldsymbol{r}_i\}_{i\in[\tau]}$ and openings L.
5. $\mathcal{S}$ outputs $(\{\boldsymbol{t}_i\}_{i\in[\tau]}, \pi_{\text{AEx}}, L, \pi_{\text{Shuf}})$.

Given such a string $(\{\boldsymbol{t}_i\}_{i\in[\tau]}, \pi_{\text{AEx}}, L, \pi_{\text{Shuf}})$ from $\mathcal{S}$ as well as ciphertext vector $\{\boldsymbol{c}_i\}_{i\in[\tau]}$ any third party $\mathcal{V}$ can now run the following algorithm Π_{MixV} to verify the mix step:

1. Run the verification algorithm of Π_{AExV} for π_{AEx} on inputs $\boldsymbol{A}''$, $\{\boldsymbol{t}_i\}_{i\in[\tau]}$ and B. If the verification fails, then output 0.
2. For all $i \in [\tau]$ set $[\![(\hat{u}_i, \hat{v}_i)]\!] = [\![(u_i, v_i)]\!]_{\mathbf{0}} + \boldsymbol{t}_i$ where $(u_i, v_i) = \boldsymbol{c}_i$.
3. Run the verification algorithm of Π^ℓ_{ShufV} for π_{Shuf} on input $\{[\![(\hat{u}_i, \hat{v}_i)]\!]\}_{i\in[\tau]}, L$. If the verification fails, then output 0.
4. Output 1.

The following theorems refer to definitions of correctness, soundness and honest-verifier zero-knowledge given in Section 2.7. The protocol Π_{Mix} is essentially a re-randomization and shuffle of a set of ciphertexts augmented with some commitments and zero-knowledge proofs, carefully composed to give security.

In the following theorems, define the noise bound B_{Dec} to be the maximum level of noise in a ciphertexts $\boldsymbol{c}'_i = \mathsf{Enc}(\mathsf{pk}, m'_i)$ when the randomness $r'_i, e'_{i,1}, e'_{i,2}$ used to create the ciphertexts is bounded by B_∞ and m'_i is bounded by p in the ℓ_∞ norm. Let B_{Dec} satisfy $2B_{\mathsf{Dec}} < \lfloor q/2 \rfloor$.

Theorem 8 (Correctness). *Let input ciphertexts have noise bounded by B_{Dec}, and let the total noise added in Π_{Mix} be bounded by B_{Dec}. Suppose the protocols Π_{AEx} and Π^ℓ_{Shuf} are complete. Then the mixing protocol is correct.*

We sketch the argument. Since $2B_{\mathsf{Dec}} < \lfloor q/2 \rfloor$, it follows that decryption is correct. Furthermore, since Π_{AEx} and Π^ℓ_{Shuf} are complete, the arguments will be accepted, which means that the mixing proof will be accepted.

Theorem 9 (Special Soundness). *Let $\mathcal{E}_1$ be a knowledge extractor for the protocol Π_{AEx} with success probability ϵ_1 and let $\mathcal{E}_2$ be a knowledge extractor for the protocol Π^ℓ_{Shuf} with success probability ϵ_2. Then we can construct a knowledge extractor $\mathcal{E}_0$ for the mixing protocol that succeeds with probability $\epsilon_0 \geq \epsilon_1 + \epsilon_2$. The runtime of $\mathcal{E}_0$ is essentially the same as the runtime of $\mathcal{E}_1$ and $\mathcal{E}_2$.*

We sketch the argument. The main observation is that it is enough that either of the extractors $\mathcal{E}_1$ and $\mathcal{E}_2$ succeeds.

If the extractor $\mathcal{E}_1$ succeeds, we are able to extract τ randomness vectors $\boldsymbol{r}_i'$ bounded by B_∞, which gives us the randomness for both the commitments and ciphertexts used in the protocol. Then we can directly extract the permutation π by inspection, and hence, we have an extractor $\mathcal{E}_0$ for the mixing protocol Π_{Mix}.

If the extractor $\mathcal{E}_1$ succeeds, we are able to extract both the permutation π and τ randomness vectors $\boldsymbol{r}_i$ used in the commitments and, indirectly, also the committed randomness used to create the encryption of 0. Hence, we have an extractor $\mathcal{E}_0$ for the mixing protocol Π_{Mix}.

If neither of these strategies works, we have an attacker against the binding property of the commitment scheme.

Theorem 10 (Honest-Verifier Zero-Knowledge). *Suppose the protocol Π_{AEx} and Π^ℓ_{Shuf} are honest-verifier zero-knowledge, that* `Com` *is hiding and that* `Enc` *is CPA secure. Then there exists a simulator for the mixing protocol such that for any distinguisher $\mathcal{A}_0$ for this simulator with advantage ϵ_0, there exists an adversary $\mathcal{A}_3$ against hiding for the commitment scheme with advantage ϵ_3, an adversary $\mathcal{A}_4$ against CPA security for the encryption scheme with advantage ϵ_4, and distinguishers $\mathcal{A}_1$ and $\mathcal{A}_2$ for the simulators for Π_{AEx} and Π^ℓ_{Shuf}, respectively, with advantages ϵ_1 and ϵ_2, such that $\epsilon_0 \leq \epsilon_1+\epsilon_2+\epsilon_3+\epsilon_4$. The runtime of $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3, \mathcal{A}_4$ are essentially the same as $\mathcal{A}_0$.*

We sketch the argument. The simulator is given the set of messages encrypted by the input ciphertexts. The simulator simulates the zero-knowledge proofs Π_{AEx} and Π^ℓ_{Shuf} using the appropriate simulators. It replaces the commitments to the ciphertexts (u_i', v_i') by random commitments and the output ciphertexts by fresh ciphertexts to the correct messages.

The claim about the simulator follows from a hybrid argument. We begin with the verifiable shuffle protocol.

First, we replace the Π_{Shuf} arguments by simulated arguments, which gives us a distinguisher $\mathcal{A}_2$ for the Π_{Lin} honest verifier simulator.

Second, we replace the Π_{AEx} arguments by simulated arguments, which gives us a distinguisher $\mathcal{A}_1$ for the Π_{AEx} honest verifier simulator.

Third, we replace the commitments to ciphertexts by random commitments, which gives us an adversary $\mathcal{A}_3$ against hiding for the commitment scheme.

Fourth, we replace the output ciphertexts by fresh ciphertexts, which gives us an adversary $\mathcal{A}_4$ against CPA security.

After the changes, we are left with the claimed simulator for the actively secure protocol, and the claim follows.

5 Verifiable Distributed Decryption

In this section we provide a construction for verifiable distributed decryption. We combine the distributed decryption protocol from Section 3.1 with zero-knowledge proofs to achieve an actively secure decryption protocol. In this protocol, the set of ciphertexts is given to a number of decryption servers each holding a share of the secret key. All of the servers compute a partial decryption of each ciphertext. Finally, they use noise drowning to hide the secret key and their published shares are added and rounded to output the plaintext. The partial decryption is a linear operation, and we prove correctness using zero-knowledge proofs of linear relations from Section 3.3, and we use the amortized proof from Section 3.5 to prove that the noise is bounded to ensure correct decryption. Both proofs are computed using the commitment scheme from Section 3.2. We also provide an optimistic distributed decryption protocol given in Appendix A which is secure if not all decryption servers are colluding simultaneously.

5.1 The Actively Secure Protocol

Public parameters. Let the ring R_q, the bounded distribution $\mathtt{D}$ over R_q, the statistical security parameter $\mathtt{sec}$ and error bounds $B_{\mathtt{Com}}$ and $B_{\mathtt{Dec}}$ be public system information, together with the plaintext modulus p for the encryption system. Let $\boldsymbol{A}$ be the public commitment matrix for message size $\ell = 1$ over R_q.

- $\mathtt{KeyGen}_A(1^\lambda, \xi)$:
  1. Compute $(\mathsf{pk}, \mathsf{sk}, s_1, \ldots, s_\xi) \leftarrow \mathtt{KeyGen}(1^\lambda, \xi)$ as in the passive protocol.
  2. For each $j \in [\xi]$ compute $(\llbracket s_j \rrbracket, \boldsymbol{d}_j) \leftarrow \mathtt{Com}(s_j)$.
  3. Output $\mathsf{pk}_A = (\mathsf{pk}, \llbracket s_1 \rrbracket, \ldots, \llbracket s_\xi \rrbracket)$, $\mathsf{sk}_A = \mathsf{sk}$ and $\mathsf{sk}_{A,j} = (s_j, \boldsymbol{d}_j)$.

- $\mathtt{Enc}_A$ and $\mathtt{Dec}_A$ works just like the original $\mathtt{Enc}$ and $\mathtt{Dec}$ in the passively secure threshold encryption scheme, ignoring additional information in pk_A.

- $\mathtt{DistDec}(\mathsf{sk}_{A,j}, \{c_i\}_{i\in[\tau]})$ where $c_i = (u_i, v_i)$:
  1. For each $i \in [\tau]$ do the following. First, compute $m_{i,j} = s_j u_i$.
  2. Sample uniform noise $E_{i,j} \leftarrow R_q$ such that $\|E_{i,j}\|_\infty \leq 2^{\mathtt{sec}}(B_{\mathtt{Dec}}/p\xi)$.
  3. Compute the message decryption share $t_{i,j} = m_{i,j} + pE_{i,j}$.
  4. Compute $(\llbracket E_{i,j} \rrbracket, \boldsymbol{r}''_{i,j}) \leftarrow \mathtt{Com}(E_{i,j})$ and use the Π_{LIN}-protocol to compute a proof for the linear relation $t_{i,j} = s_j u_i + pE_{i,j}$ by computing
  $$\pi_{L_{i,j}} \leftarrow \Pi_{\text{LIN}}(((s_j, \boldsymbol{r}_j), (E_{i,j}, \boldsymbol{r}''_{i,j})); (\llbracket s_j \rrbracket, \llbracket E_{i,j} \rrbracket, t_{i,j}), (u_i, p)).$$
  5. The commitment to $E_{i,j}$ is of the form
  $$\llbracket E_{i,j} \rrbracket = \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{a}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{0}^n & 1 & \boldsymbol{a}_{2,2} \end{bmatrix} \cdot \boldsymbol{r}''_{i,j} + \begin{bmatrix} 0 \\ E_{i,j} \end{bmatrix}$$
  $$= \underbrace{\begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{a}_{1,1} & \boldsymbol{A}_{1,2} & 0 \\ \boldsymbol{0}^n & 1 & \boldsymbol{a}_{2,2} & 1 \end{bmatrix}}_{\boldsymbol{A}''} \begin{bmatrix} \boldsymbol{r}''_{i,j} \\ E_{i,j} \end{bmatrix},$$
  where $\left\|\boldsymbol{r}''_{i,j}\right\|_\infty \leq B_{\mathtt{Com}}$ is the randomness used in the commitments. Run the Π_{ANEx}-protocol, denoted as $\Pi_{\text{ANEx}}(\{(E_{i,j}, \boldsymbol{r}''_{i,j}\}_{i\in[\tau]}); (\boldsymbol{A}'', \{\llbracket E_{i,j} \rrbracket\}_{i\in[\tau]}))$, and obtain the amortized zero-knowledge proof of knowledge $\pi_{\text{ANEx}_j} = (\boldsymbol{C}'', \boldsymbol{Z}'')$ with binary challenge matrix $\boldsymbol{C}''$.
  6. Output $\mathtt{ds}_j = (\{t_{i,j}\}_{i=1}^{\tau}, \pi_{\mathcal{D}_j})$ where $\pi_{\mathcal{D}_j} = (\{\llbracket E_{i,j} \rrbracket\}_{i=1}^{\tau}, \{\pi_{L_{i,j}}\}_{i=1}^{\tau}, \pi_{\text{ANEx}_j})$.

– $\texttt{Comb}_A(\{c_i\}_{i=1}^{\tau}, \{\texttt{ds}_j\}_{j\in[\xi]})$:
1. Parse $\texttt{ds}_j$ as $(\{t_{i,j}\}_{i=1}^{\tau}, \pi_{\mathcal{D}_j})$.
2. Verify the proofs $\pi_{L_{i,j}}$: $\quad 0 \vee 1 \leftarrow \mathit{\Pi}_{\text{LinV}}([\![s_j]\!], [\![E_{i,j}]\!], t_{i,j}), (u_i, p); \pi_{L_{i,j}})$.
3. Verify the proofs π_{ANEx_j}: $\quad 0 \vee 1 \leftarrow \mathit{\Pi}_{\text{ANExV}}(\boldsymbol{A}'', \{[\![E_{i,j}]\!]\}_{i\in[\tau]}; \pi_{\text{ANEx}_j})$.
4. If any verification protocol returned 0 then output $\perp$. Otherwise compute

$$m_i = (v_i - t_i \mod q) \mod p, \text{ where } t_i = t_{i,1} + \cdots + t_{i,\xi} \text{ for } i = 1, \ldots, \tau,$$

and output the set of messages $m_1, \ldots, m_\tau$.

Remark 1. The randomness $\boldsymbol{r}''_{i,j}$ has much smaller ℓ_∞ norm than $E_{i,j}$, and hence, we will run the $\mathit{\Pi}_{\text{ANEx}}$ protocol with small standard deviation σ_{ANEx} for rows 1 to k, while row $k+1$ will have large standard deviation $\hat{\sigma}_{\text{ANEx}}$, as noted in 3.5.

The following theorems refer to definitions of threshold correctness, threshold verifiability and distributed decryption simulatability given in Section 2.5. It is important to understand that the protocol is essentially a passively secure distributed decryption protocol augmented with some commitments and some zero-knowledge proofs, carefully composed to give active security.

In the following three theorems, let the noise bounds $B_{\texttt{Dec}}$ and $\hat{B}_{\text{ANEx}}$ satisfy $(1 + B_{\texttt{Dec}}) \cdot 2^{\sec} < 2\hat{B}_{\text{ANEx}} < \lfloor q/2 \rfloor$.

Theorem 11 (Threshold Correctness). *Let ciphertexts have noise bounded by $B_{\texttt{Dec}}$, and let the total noise added in* $\texttt{DistDec}$ *be bounded by $2^{\sec}B_{\texttt{Dec}}$. Suppose the passively secure protocol is threshold correct and the protocols $\mathit{\Pi}_{\text{Lin}}$ and $\mathit{\Pi}_{\text{ANEx}}$ are complete. Then the actively secure protocol is threshold correct.*

We sketch the argument. Since $B_{\texttt{Dec}} + 2^{\sec}B_{\texttt{Dec}} < q/2$, it follows that decryption is correct. Furthermore, since $(1 + B_{\texttt{Dec}}) \cdot 2^{\sec} < 2\hat{B}_{\text{ANEx}} < q/2$ and $\mathit{\Pi}_{\text{Lin}}$ and $\mathit{\Pi}_{\text{ANEx}}$ are complete, the arguments will be accepted, which means that the decryption proof will be accepted.

Theorem 12 (Threshold Verifiability). *Let $\mathcal{A}_0$ be an adversary against threshold verifiability for the actively secure protocol with advantage ϵ_0. Then there exists adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ against soundness for $\mathit{\Pi}_{\text{Lin}}$ and $\mathit{\Pi}_{\text{ANEx}}$, respectively, with advantages ϵ_1 and ϵ_2, such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2$. The runtime of $\mathcal{A}_1$ and $\mathcal{A}_2$ are essentially the same as the runtime of $\mathcal{A}_0$.*

We sketch the argument. We need only consider ciphertexts with noise bounded by $B_{\texttt{Dec}}$, so we may assume that the noise in any particular ciphertext is bounded by $B_{\texttt{Dec}}$.

If the decryption is incorrect for a particular ciphertext, then for some j no relation $t_{i,j} = s_j u_i + pE_{i,j}$ holds for an $E_{i,j}$ of norm at most $2\hat{B}_{\text{ANEx}}$.

This can happen in two ways: Either the argument for the linear combination of the commitments to $E_{i,j}$ and s_j is incorrect, or the bound on $E_{i,j}$ is incorrect.

In the former case, we trivially get an adversary $\mathcal{A}_1$ against soundness for $\mathit{\Pi}_{\text{Lin}}$. Similar for the latter case and $\mathit{\Pi}_{\text{ANEx}}$.

Theorem 13 (Distributed Decryption Simulatability). *Suppose the passively secure protocol is simulatable and Π_{LIN} and Π_{ANEx} are honest-verifier zero-knowledge. Then there exists a simulator for the actively secure protocol such that for any distinguisher $\mathcal{A}_0$ for this simulator with advantage ϵ_0, there exists an adversary $\mathcal{A}_4$ against hiding for the commitment scheme*[5]*, with advantage ϵ_4, and distinguishers $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ for the simulators for the passively secure protocol, Π_{LIN} and Π_{ANEx}, respectively, with advantages ϵ_1, ϵ_2 and ϵ_3, such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4$. The runtime of $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$ are essentially the same as the runtime of $\mathcal{A}_0$.*

We sketch the argument. The simulator simulates the arguments and the passively secure distributed decryption algorithm, using the appropriate simulators. Also, it replaces the commitments to the noise $E_{i,j}$ by random commitments.

The claim about the simulator follows from a straight-forward hybrid argument. We begin with the distributed decryption algorithm.

First, we replace the Π_{LIN} arguments by simulated arguments, which gives us a distinguisher $\mathcal{A}_2$ for the Π_{LIN} honest verifier simulator.

Second, we replace the Π_{ANEx} arguments by simulated arguments, which gives us a distinguisher $\mathcal{A}_3$ for the Π_{ANEx} honest verifier simulator.

Third, we replace the commitments to the noise $E_{i,j}$ by random commitments, which gives us an adversary $\mathcal{A}_4$ against hiding for the commitment scheme.

Fourth, we replace the passively secure distributed decryption algorithm by its simulator, which gives us a distinguisher $\mathcal{A}_1$ for the simulator.

After the four changes, we are left with the claimed simulator for the actively secure protocol, and the claim follows.

We refer to Section 7 for sizes, parameters and a concrete instantiation of the distributed decryption, combining it with the mix-net described in Section 4.

6 A Cryptographic Voting System

Our voting protocol follows a fairly natural design paradigm of mixing and threshold decryption. Common voting scheme security definitions such as [BCG+15] do not model shuffles and distributed decryption. Following other works such as e.g. [Scy, Gjø11] we therefore define *ad hoc* security definitions. The high-level architecture for the counting phase of our protocol is shown in Figure 5.

We follow the standard approach for voting system analysis, which is to consider a voting system to be fairly simple cryptographic protocol built on top of a *cryptographic voting scheme.*

The *verifiable* cryptographic voting scheme *with return codes*, *shuffles* and *distributed decryption* is described in Appendix B. It uses our shuffle and verifiable decryption as described previously as well as other primitives. We now explain how our voting system can be described as a simple protocol on top of

[5] A more careful argument could allow us to dispense with this adversary. We have opted for a simpler argument, since the commitment scheme is also used elsewhere.

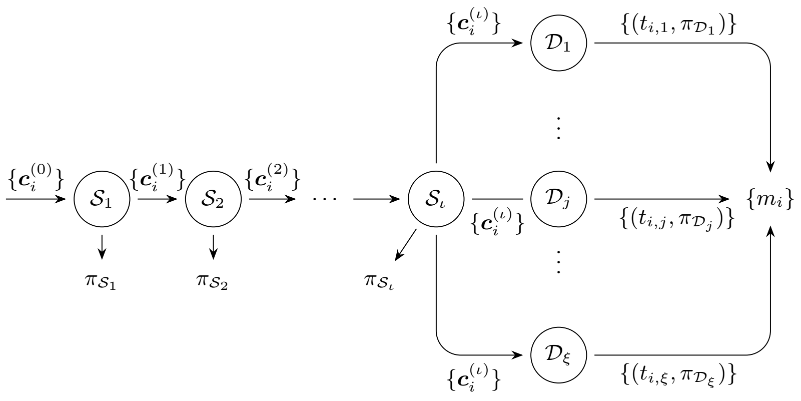

Fig. 5. The high level counting phase of our voting protocol. Each shuffle server $\mathcal{S}_k$ receives a set of ciphertexts $\{c_i^{(k-1)}\}$, shuffles them, and outputs a new set of ciphertexts $\{c_i^{(k)}\}$ and a proof $\pi_{\mathcal{S}_k}$. When all shuffle proofs are verified, each decryption server $\mathcal{D}_j$ partially decrypts every ciphertext and outputs the partial decryptions $\{t_{i,j}\}$ together with a proof of correctness $\pi_{\mathcal{D}_j}$. All votes can be reconstructed to $\{m_i\}$ from the partial decryptions. The full voting protocol also includes proofs of known messages from voters and a return code protocol for verifiability, see details in the Appendix.

this cryptographic voting scheme. We define security notions for this cryptosystem, sketch the security proof, and then informally discuss the voting protocol's security properties in terms of the cryptosystem's security in the Appendix.

6.1 Voting Protocol

The voting protocol requires a *trusted set of players* to run setup and registration, a set of *voters* $\mathcal{V}_i$ and their *computers* P_i, a *ballot box* $\mathcal{B}$, a *return code generator* $\mathcal{R}$, a collection of *shuffle servers* $\mathcal{S}_k$, a collection of *decryption servers* $\mathcal{D}_j$ and one or more *auditors* $\mathcal{A}$.

In the *setup phase*, a trusted set of players runs the setup algorithm. The key generation can either be done in a trusted fashion, or in a distributed fashion using the protocol by Rotaru *et al.* [RST+22] to get an actively secure robust threshold sharing of the secret decryption key. The derived public parameters are given to every participant, while the decryption key shares are given to the decryption servers $\mathcal{D}_j$. The code key is given to the return code generator $\mathcal{R}$, who will use the key to derive so-called *return codes* [CES02,HR16] that are sent to the voter. (As detailed in the Appendix, these codes are human-verifiable and can allow the voter to detect a cheating computer tampering with ballots.)

In the *registration phase*, a set of trusted players run the register algorithm to generate verification and casting keys for each voter $\mathcal{V}_i$, making every verification key public and giving the voter casting key to the voter's computer. Then the return code generator chooses a PRF-key, and a set of trusted players compute the return code table. The voter gets the return code table.

In the *casting phase*, each voter $\mathcal{V}_i$ instructs its computer P_i which ballot to cast. The computer runs the casting algorithm, signs the encrypted ballot and the ballot proof on the voter's behalf, and sends the encrypted ballot, the ballot proof and the signature to the ballot box $\mathcal{B}$. The ballot box $\mathcal{B}$ sends the encrypted ballot, the ballot proof and the signature to the return code generator $\mathcal{R}$, who runs the code algorithm. It uses the result to compute the return code and sends it to the voter's phone $\mathcal{F}_i$, which sends it on to the voter $\mathcal{V}_i$.

Both the ballot box and the return code generator will verify the voter's signature. After sending the return code, the return code generator countersigns the encrypted ballot, the ballot proof and the voter's signature, and sends the countersignature to the ballot box, which in turns sends the countersignature to the voter's computer. The computer verifies the countersignature and only then accepts, showing the encrypted ballot, the ballot proof, the signature and the countersignature to the voter, which constitutes the voter's *receipt*.

The voter $\mathcal{V}_i$ accepts the ballot as cast if and only if the computer accepts with a receipt, and the voter's phone shows a return code such that the pair is in the return code table.

In the *counting phase*, the ballot box $\mathcal{B}$ and the return code generator $\mathcal{R}$ send the encrypted ballots, ballot proofs and voter signatures they have seen to the auditor $\mathcal{A}$ as well as every decryption server. The ballot box $\mathcal{B}$ then sorts the list of encrypted ballots $\{\boldsymbol{c}_i\}$ and sends this to the first shuffle server $\mathcal{S}_1$ and every decryption server. In the event that some voter has cast more than one ballot, only the encrypted ballot seen last is included in this list.

The shuffle servers $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_\iota$ use the shuffle algorithm on the input encrypted ballots, passing the shuffled and re-encrypted ballots to the next shuffle server. They also pass the shuffled re-encrypted ballots and the shuffle proof to the auditor and every decryption server.

Each decryption server verifies that the data from $\mathcal{B}$ and $\mathcal{R}$ is consistent (similar to the auditor $\mathcal{A}$), and that every shuffle proof verifies. Only then they run the distributed decryption algorithm with their decryption key share and send their partial decryption shares of each ballot as well as proofs of correct decryption to the auditor.

If the data is consistent (that is, the same encrypted ballots, ballot proofs and signatures appear in the same order in the data from both $\mathcal{B}$ and $\mathcal{R}$, and the signatures and the ballot proofs verify), the auditor $\mathcal{A}$ approves.

The auditor verifies the data from $\mathcal{B}$ and $\mathcal{R}$ (the same encrypted ballots, ballot proofs and signatures appear in the same order in the two data sets, and the voter signatures and the ballot proofs verify), that the encrypted ballots received by the first shuffle are consistent with the data from $\mathcal{B}$ and $\mathcal{R}$, that every shuffle proof verifies, and then runs the combining algorithm on the received ballot decryption shares. If the combining algorithm accepts then the auditor accepts, otherwise it rejects. Finally, $\mathcal{A}$ outputs the complete list of messages received, including the public key material, as its transcript.

There is a verification algorithm that takes as input a transcript, a result and optionally a receipt, and either accepts or rejects. The verification algorithm simply runs the auditor with the public key material and the messages listed in the transcript, and checks if the auditor's result matches the input result. If a receipt is present, it also verifies the countersignature and the voters' signatures in the receipt, that the encrypted ballot, the ballot proof and the voters' signatures are present in the ballot box data set, and that the encrypted ballots are present in the first shuffle server's input.

This concludes the description of the voting protocol in terms of a verifiable cryptographic voting scheme with return codes.

Note that there are many variations of this protocol. It can be used without return codes, simply by omitting return codes. Also, depending on the exact setting and security required, the return code generator can be merged with the ballot box.

Many comparable schemes are phrased in terms of an ideal (web) bulletin board, where every player posts their messages. Implementing a bulletin board is tricky in practice, so instead we have described the scheme as a conventional cryptographic protocol passing messages via some network.

It is also worth noting that for our concrete scheme anyone can redo the auditor's work (since no secret key material is involved) by running the verification algorithm (and parts of the code algorithm) on the public data, making the voting protocol (universally) verifiable.

7 Performance

7.1 Size of the Submission Phase

Each BGV ciphertexts are of size $2N \log_2 q$ bits. We assume that the set of input-ciphertexts to the mixing network are honestly generated. We can ensure this by using e.g. the exact proofs by Beullens [Beu20] or Baum and Nof [BN20], the efficient range proofs by Attema *et al.* [ALS20], or the new techniques from Lyubashevsky *et al.* [LNS21,ENS20] to prove that the noise is bounded and that the user knows the message.

7.2 Size of the Mixing Network

Let ι denote the number of mixing servers and let τ be the number of ciphertexts. Each ciphertext consists of two elements in R_q, where each element can be represented by $N \log_2 q$ bits. The transcript of the mixing phase will contain ι sets of τ new ciphertexts, and is of size $2\iota\tau N \log_2 q$ bits.

For each shuffle the servers must provide a proof of shuffle and an amortized proof of shortness. Both proofs prove a relation about commitments to the randomization factors added to the old ciphertexts to get the new ciphertexts, and each commitment is of size $(n+2)N \log_2 q$ bits.

The shuffle proof consists of τ commitment of size $(n+1)N \log_2 q$ bits, τ ring elements of size $N \log_2 q$ bits and a proof of linearity per ciphertext. The proof

Parameter	Explanation	Constraints
λ	Security parameter	At least 128 bits
N	Degree of polynomial $X^N + 1$ in R	N a power of two
p	Plaintext modulus	p a small prime
q	Ciphertext and commitment modulus	Prime $q = 1 \mod 2N$ s.t. $\max\lVert v - su \rVert \ll q/2$
k	Width (over R_q) of commitment matrix	
n	Height (over R_q) of commitment matrix	
ν	Maximum l_1-norm of elements in $\mathcal{C}$	
$\mathtt{D}$	Bounded distribution for noise in ciphertexts	Chosen to be the uniform ternary distribution
σ_{C}	Standard deviation for one-time commitments	Chosen to be $\sigma_{\mathrm{C}} = 0.954 \cdot \nu \cdot \beta_\infty \cdot \sqrt{kN}$
$\hat{\sigma}_{\mathrm{C}}$	Standard deviation for reusable commitments	Chosen to be $\hat{\sigma}_{\mathrm{C}} = 22 \cdot \nu \cdot \beta_\infty \cdot \sqrt{kN}$
σ_{ANEx}	Standard deviation for the one-time amortized proof	Chosen to hide the commitment randomness $\boldsymbol{r}''_{i,j}$
$\hat{\sigma}_{\mathrm{ANEx}}$	Standard deviation for the one-time amortized proof	Chosen to hide the noise-drowning values $E_{i,j}$
$\mathcal{C}$	Challenge space	$\mathcal{C} = \{c \in R_p \mid \lVert c \rVert_\infty = 1, \lVert c \rVert_1 = \nu\}$
$\bar{\mathcal{C}}$	The set of differences $\mathcal{C} - \mathcal{C}$ excluding 0	$\bar{\mathcal{C}} = \{c - c' \mid c \neq c' \in \mathcal{C}\}$
S_{β_∞}	Set of elements of ∞-norm at most β_∞	$S_{\beta_\infty} = \{x \in R_p \mid \lVert x \rVert_\infty \leq \beta_\infty\}$
$\hat{n}$	Dimension of proof in Π_{ANEx}	$\hat{n} \geq \lambda + 2$
ι, ξ	Number of shuffle- and decryption-servers	
τ	Total number of messages	For soundness we need $(\tau^\delta + 1)/\lvert R_q \rvert < 2^{-128}$

Table 1. System parameters and constraints.

of linearity is of size $2(k-n)N\log_2(6\sigma_{\mathrm{C}})$ bits. The shuffle proof is then of size $((n+2)N\log_2 q + 2(k-n)N\log_2(6\sigma_{\mathrm{C}}))\tau$ bits.

The amortized proof of shortness depends on the bound, in our case $B = 1$, but also on the ratio between the number of commitments and the size of the modulus. We denote the proof by π_{AEx}, and discuss it in more detail later based on concrete parameters. The total bit-size of the mixing is

$$\iota((2n+6)N\log_2 q + 2(k-n)N\log_2(6\sigma_{\mathrm{C}}))\tau + \iota|\pi_{\mathrm{AEx}}|.$$

7.3 Size of the Distributed Decryption

Let ξ denote the number of decryption servers and let τ be the number of ciphertexts. Each partial decryption consists of one element from R_q, which means that the output of the decryption is of size $\xi\tau N\log_2 q$ bits.

Additionally, each decryption server outputs a commitment to the added noise and a proof of linearity per ciphertext, and an amortized proof of shortness for all the added noise values. Also, each server has a public commitment of their decryption key-share to be used in the proof of linearity. Each commitment is of size $(n+1)N\log_2 q$ bits, and each proof of linearity is of size $(k-n)N(\log_2(6\sigma_{\mathrm{C}}) + \log_2(6\hat{\sigma}_{\mathrm{C}})$ bits (because the partial decryption is given in the clear and one commitment is re-used in all equations). Finally, each of the amortized proofs is of size $k\hat{n}N\log_2(6\sigma_{\mathrm{ANEx}}) + \hat{n}\log_2(6\hat{\sigma}_{\mathrm{ANEx}})$ bits because of the different norms of the secret values as noted in Remark 1. As the bounds in the amortized proof depends on the number of commitments in the statement, we compute batched proofs of N equations at once to control the growth.

The total size of the distributed decryption is

$$\xi((n+2)N\log_2 q + (k-n)N(\log_2(6\sigma_{\text{C}}) + \log_2(6\hat{\sigma}_{\text{C}})) \\ + k\hat{n}\log_2(6\sigma_{\text{ANEx}}) + \hat{n}\log_2(6\hat{\sigma}_{\text{ANEx}}))\tau \text{ bits.}$$

7.4 Concrete Parameters and Total Size

Standard deviation. We let the success probability of each of the zero-knowledge protocols to be $1/M \approx 1/3$. The algorithm in Section 2.2 is used for rejection sampling. We will use the following parameters, where we note that the commitments used in the shuffle and in the amortized proofs are only used once, while the proof of linearity in the decryption protocol depends on a commitment to the secret key-share each time. However, that is the only part that is reused, and we can use a smaller standard deviation for the other commitment.

The proofs of linearity have two terms, which means that each of them must have a success probability of $1/\sqrt{3}$. This gives $\sigma_{\text{C}} = 0.954\nu\beta_\infty\sqrt{kN}$. For the re-usable commitments we get $\hat{\sigma}_{\text{C}} = 22\nu\beta_\infty\sqrt{kN}$. The amortized proof also have two checks, and we get standard deviation $0.954\|\boldsymbol{S}'\boldsymbol{C}'\|_2$, where σ_{ANEx} and $\hat{\sigma}_{\text{ANEx}}$ are depending on the norm of the elements in the rows of $\boldsymbol{S}'$.

For the encryption, we let $\mathtt{D}$ be the ternary distribution over R_q, where each polynomial has coefficients in $\{-1,0,1\}$ sampled uniformly at random. We let the commitment randomness be bounded by $B_{\mathtt{Com}} = 1$.

Bounding the noise. To be able to choose concrete parameters for the mix-net, we need to estimate how much noise that is added to the ciphertexts through the two stages of the protocol: 1) the shuffle phase, and 2) the decryption phase. Each part of the system contributes the following amount of noise to the ciphertexts:

- Original ciphertext: $B_{\mathtt{Start}} = p(\|er\|_\infty + \|e_{i,2}\|_\infty + \|-e_{i,1}s\|_\infty) + \|m\|_\infty$.
- Additional noise per shuffle: $B_{\text{SHUF}} = p(\|er'\|_\infty + \|e'_{i,2}\|_\infty + \|-e'_{i,1}s\|_\infty)$.
- Additional noise in partial decryption: $B_{\mathtt{DistDec}} = p\xi\|E'_{i,j}\|_\infty \leq 2^{\mathsf{sec}}B_{\mathtt{Dec}}$,

where $B_{\mathtt{Dec}} = B_{\mathtt{start}} + \iota B_{\text{SHUF}}$ is the upper bound of the noise added before the decryption phase. This means that we have the following bounds on each of the noise-terms above, when using ternary noise:

$$\|e\|_1 \leq N, \quad \|r\|_\infty \leq 1, \quad \|e_{i,2}\|_\infty \leq 1, \quad \|e_{i,1}\|_1 \leq N,$$
$$\|s\|_\infty \leq 1, \quad \|r'\|_\infty \leq 1, \quad \|e'_{i,2}\|_\infty \leq 1, \quad \|e'_{i,1}\|_1 \leq N.$$

Using the bounds from Section 2, we get upper bounds:

$$B_{\mathtt{Start}} = p(2N+1) + \lceil (p-1)/2 \rceil, \quad B_{\text{SHUF}} = p(2N+1),$$

which for ι shuffles gives us

$$B_{\mathtt{Dec}} = (\iota+1)p(2N+1) + \lceil (p-1)/2 \rceil.$$

Finally, we need to make sure that $B_{\texttt{Dec}} + B_{\texttt{DistDec}} < q/2$, where $B_{\texttt{DistDec}} = 2p\xi\hat{B}_{\text{ANEx}}$ because of the soundness slack of the amortized proof of bounded values from Section 3.5. A honestly generated value $E_{i,j}$ is bounded by $2^{\texttt{sec}}(B_{\texttt{Dec}}/p\xi)$, but the proof can only guarantee that the values are shorter than some larger bound $2\hat{B}_{\text{ANEx}}$ (following [BBC+18, Lemma 3]) that depends on the number of equation in the statement. Define $\boldsymbol{S}''$ to be the first k rows of $\boldsymbol{S}'$ and define $\boldsymbol{S}'''$ to be the last row of $\boldsymbol{S}'$. For batches of N equations we then get that:

$$\begin{aligned} B_{\text{ANEx}} &\leq \sqrt{2N} \cdot \sigma_{\text{ANEx}} \leq \sqrt{2N} \cdot 0.954 \cdot \max\left\|\boldsymbol{S}''\boldsymbol{C}'\right\|_2 \\ &\leq 1.35 \cdot \sqrt{N} \cdot \max\left\|\boldsymbol{S}''\right\|_1 \cdot \max\left\|\boldsymbol{C}'\right\|_\infty \\ &\leq 1.35 \cdot k \cdot \sqrt{N} \cdot N \cdot B_{\texttt{Com}}, \end{aligned}$$

and, similarly,

$$\hat{B}_{\text{ANEx}} \leq \sqrt{2N} \cdot \hat{\sigma}_{\text{ANEx}} \leq 1.35 \cdot \sqrt{N} \cdot N \cdot \|E_{i,j}\|_\infty,$$

with B_{ANEx} for rows 1 to k of $\boldsymbol{Z}$ and $\hat{B}_{\text{ANEx}}$ for the last.

N	p	q	$\texttt{sec}$	ι	ξ	n	k	ν	$B_{\texttt{Com}}$	$\hat{n}$	σ_{C}	$\hat{\sigma}_{\text{C}}$	σ_{ANEx}	$\hat{\sigma}_{\text{ANEx}}$	$\hat{B}_{\text{ANEx}}$
4096	2	$\approx 2^{78}$	40	4	4	1	$\ell + 2$	36	1	130	$\approx 2^{12}$	$\approx 2^{16.5}$	$\approx 2^{13.5}$	$\approx 2^{66}$	$\approx 2^{72.5}$

Table 2. Concrete parameters estimated for $\lambda \approx 168$ bits of security using the LWE-estimator by Albrecht *et al.* [APS15].

We fix plaintext modulus $p = 2$, statistical security parameter $\texttt{sec} = 40$, and need $N = 4096$ when q is large to provide proper security. This allows for votes of size 4096 bits, which should be a feasible size for real-world elections. We let the number of shuffle and decryption servers be $\iota = \xi = 4$. It follows that $B_{\texttt{Dec}} < 2^{17}$ and $B_{\texttt{DistDec}} < 2^{76.5}$. We then set $q \approx 2^{78}$, and verify that

$$\max_{i \in [\tau]} \|v_i - su_i\| < 2 \cdot (2^{17} + 2^{76.5}) < q.$$

Exact amortized proof. Finally, we must decide on parameters for the exact proof of shortness from Section 3.4. The soundness of the protocol depends on the ration between the number of equations and the size of the modulus, see Lemma 1. We choose to compute the proof in batches of size N instead of computing the proof for all τ commitments at once. Then we get $18N/(q - N) \approx 2^{-62}$, and hence, we must compute each proof twice in parallel to achieve negligible soundness. Furthermore, we choose $k \approx 2^{20}, l \approx 2^{20.3}, \eta = 325$ to keep the soundness $\approx 2^{-62}$. The total size of π_{AEx}, by instantiating 2, is $\approx 20\tau$ KB.

Total size. We give a complete set of parameters in Table 2, and the concrete sizes of each part of the protocol in Table 3. Each voter submit a ciphertext size approximately 80 KB. The size of the mix-net, including ciphertexts, commitments, shuffle proof and proof of shortness, is approximately 370τ KB per mixing node $\mathcal{S}_i$. The size of the decryption phase, including partial decryptions, commitments, proofs of linearity and proofs of boundedness, is approximately 157τ KB per decryption node $\mathcal{D}_j$.

$\boldsymbol{c}_i^{(k)}$	$[\![R_q^\ell]\!]$	π_{Shuf}	$\pi_{L_{i,j}}$	π_{AEx}	π_{ANEx}	$\pi_{\mathcal{S}_i}$	$\pi_{\mathcal{D}_j}$
80 KB	$40(\ell+1)$ KB	150τ KB	35 KB	20τ KB	2τ KB	370τ KB	157τ KB

Table 3. Size of the ciphertexts, commitments and proofs.

7.5 Implementation

In order to estimate the efficiency of our protocols, we developed a proof-of-concept implementation to compare with previous results in the literature. Our performance figures were collected on an Intel Skylake Core i7-6700K CPU machine running at 4GHz without TurboBoost. The results can be found in Tables 4 and 5, and we intend to release the code publicly in the near future.

First, we compare performance of the main building blocks with an implementation of the shuffle proof protocol proposed in [ABG$^+$21]. That work used the FLINT library to implement arithmetic involving polynomials of degree $N = 1024$ with 56-bit coefficients, fitting a 64-bit machine word. Their parameters were not compatible with the fast Number Theoretic Transform (NTT), so a CRT decomposition to two half-degree polynomials was used instead. The code was made available, so a direct comparison is possible.

In this work, the degree is much larger ($N = 4096$) and coefficients are multi-word ($q \approx 2^{78}$), but the parameters are compatible with the NTT. We implemented polynomial arithmetic with the efficient NFLlib [ABG$^+$16] library using the RNS representation for coefficients. The arithmetic is accelerated with AVX2 instructions, especially the NTT transform and polynomial arithmetic. We observed that our polynomial multiplication is around 19 times more efficient than [ABG$^+$21] ($61,314$ cycles instead of $1,165,997$), despite parameters being considerably larger. We also employed the FLINT library for arithmetic routines not supported in NFLlib, such as polynomial division, but that incurred some non-trivial costs to convert representations between two libraries. For Gaussian sampling, we adapted COSAC [ZSS20] and adjusted the standard deviation σ accordingly.

Computing a commitment takes 0.45 ms on the target machine, which is 2x faster than [ABG$^+$21]. Opening a commitment is slower due to conversions between libraries for performing the norm test. Our implementation of BGV encryption is much faster than the 69 ms reported for verifiable encryption in [ABG+21], while decryption is improved by a factor of 7. Distributed decryption with passive security costs additional 1.51 ms per party, but the zero-knowledge proofs for active security increase the cost further. The shuffle proof performance is at 30 ms per vote, thus close to [ABG+21].

For the other sub-protocols, we benchmarked executions with $\tau = 1000$ and report the execution time amortized per vote for both prover and verifier in Table 5. In the case of Π_{AEx}, we only implement the performance-critical polynomial arithmetic and commitment scheme, since this is already representative of the overall performance. From the table, we can compute the cost of distributed decryption with active security as $(10.7 + 30 + 15.7 + 25.0) = 81.4$ ms per vote, the cost of Π_{Mix} as $(0.45 + 1009 + 15.1) = 1024$ ms and the cost of Π_{MixV} as $(20 + 16.1) = 36.1$ ms per vote. This result compares very favorably with the costs of 1.5 s and 1.49 s per vote to respectively generate/verify a proof in the lattice-based shuffle proof of [FWK21] in a Haswell processor running at approximately half the frequency. By considering space-time efficiency as a metric, our total numbers are 1.4 times higher after adjusting for clock frequency and negligible soundness, while storage overhead is much lower.

Primitive	Commit	Open	Encrypt	Decrypt	DistDec
Time	0.45 ms	2.3 ms	2.5 ms	0.83 ms	1.51 ms

Table 4. Timings for basic cryptographic operations. Numbers were obtained by computing the average of 10^4 consecutive executions of an operation measured using the cycle counter available in the platform.

Protocol	$\Pi_{\text{Lin}} + \Pi_{\text{LinV}}$	$\Pi^{\ell}_{\text{Shuf}} + \Pi^{\ell}_{\text{ShufV}}$	$\Pi_{\text{ANEx}} + \Pi_{\text{ANExV}}$	$\Pi_{\text{AEx}} + \Pi_{\text{AExV}}$
Time	$(10.7 + 15.7)\tau$ ms	$(15.1 + 16.1)\tau$ ms	$(30.0 + 25.0)\tau$ ms	$(1009 + 20)\tau$ ms

Table 5. Timings for cryptographic protocols, obtained by computing the average of 100 consecutive executions with $\tau = 1000$.

8 Concluding Remarks

We have proposed a verifiable secret shuffle of BGV ciphertexts and a verifiable distributed decryption protocol. Together, these two novel constructions are practical and solve a long-standing problem in the design of quantum-safe cryptographic voting systems.

Verifiable secret shuffles for discrete logarithm-based cryptography has seen a long sequence of incremental designs follow Neff's breakthrough construction. While individual published improvements were often fairly small, the overall improvement in performance over time was significant. We expect that our designs can be improved in a similar fashion. In particular, we expect that the size of the proofs can be significantly reduced. While it is certainly straight-forward to download a few hundred gigabytes today (compare with high-quality video streaming), many voters will be discouraged and this limits the universality of verification in practice. It therefore seems reasonable to focus further effort on reducing the size of the proofs.

The distributed decryption protocol does not have an adjustable threshold. In practice, this is not much of a problem, since the key material will be secret shared among many key holders. Only when counting starts is the key material given to the decryption servers. Key reconstruction can then be combined with a suitable distributed key distribution protocol.

Shuffles followed by distributed decryption is one paradigm for the design of cryptographic voting systems. Another possible paradigm is to use key shifting in the shuffles. This would then allow us to use a single party for decryption (though it must still be verifiable, e.g., using the protocol by Gjøsteen *et al.* [GHM+21] or by Silde [Sil22]). Key shifting can be done with many of the same techniques that we use for distributed decryption, but there seems to be difficulties in amortizing the proofs. This means that key shifting with just the techniques we use will be significantly slower and of increased size, as we would have to add an additional proof of linearity for each new ciphertext in each shuffle.

Finally, we note that our scheme and concrete instantiation using the NTT is optimized for speed, and that it is possible to slightly decrease the parameters by instantiating the encryption scheme based on the SKS^2 and DKS^∞ problems in higher dimensions k using a smaller, but still a power of 2, ring-dimension N. We leave this as future work. We also remark that lattice-based cryptography, and especially lattice-based zero-knowledge proofs such as the recent preprint by Lyubashevsky *et al* [LNP22], continuously improves the state-of-the-art, and we expect future works to improve the concrete efficiency of our protocol.

References

ABG+16. Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. NFLlib: NTT-based fast lattice library. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 341–356. Springer, Heidelberg, February / March 2016.

ABG+21. Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde, and Thor Tunge. Lattice-based proof of shuffle and applications to electronic voting. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 227–251. Springer, Heidelberg, May 2021.

Adi08. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008*, pages 335–348. USENIX Association, July / August 2008.

ALS20. Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler. Practical product proofs for lattice commitments. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 470–499. Springer, Heidelberg, August 2020.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

BBC+18. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 669–699. Springer, Heidelberg, August 2018.

BCG$^+$15. David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society Press, May 2015.

BCG$^+$17. Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 336–365. Springer, Heidelberg, December 2017.

BD10. Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 201–218. Springer, Heidelberg, February 2010.

BDL$^+$18. Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 368–385. Springer, Heidelberg, September 2018.

BEPU$^+$20. Carsten Baum, Daniel Escudero, Alberto Pedrouzo-Ulloa, Peter Scholl, and Juan Ramón Troncoso-Pastoriza. Efficient protocols for oblivious linear function evaluation from ring-LWE. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 130–149. Springer, Heidelberg, September 2020.

Beu20. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 183–211. Springer, Heidelberg, May 2020.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

BHM20. Xavier Boyen, Thomas Haines, and Johannes Müller. A verifiable and practical lattice-based decryption mix net with external auditing. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 336–356. Springer, Heidelberg, September 2020.

BHM21. Xavier Boyen, Thomas Haines, and Johannes Müller. Epoque: Practical end-to-end verifiable post-quantum-secure e-voting. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, pages 272–291. IEEE, 2021.

BLNS21. Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. More efficient amortization of exact zero-knowledge proofs for lwe. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *Computer Security – ESORICS 2021*, pages 608–627, Cham, 2021. Springer International Publishing.

BLS19. Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler. Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 176–202. Springer, Heidelberg, August 2019.

Blu84. Manuel Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1:175–193, 1984.

BN20. Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 495–526. Springer, Heidelberg, May 2020.

CES02. e-voting security study. CESG, United Kingdom, July 2002. Issue 1.2.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic LWE based E-voting scheme. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 245–265. Springer, Heidelberg, 2016.

Cha81. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

CMM19. Núria Costa, Ramiro Martínez, and Paz Morillo. Lattice-based proof of a shuffle. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *FC 2019 Workshops*, volume 11599 of *LNCS*, pages 330–346. Springer, Heidelberg, February 2019.

CP93. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993.

Dam10. Ivan Damgård. On σ-protocols, 2010. `https://cs.au.dk/~ivan/Sigma.pdf`.

DKL$^+$13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.

dLNS17. Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. Practical quantum-safe voting from lattices. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1565–1581. ACM Press, October / November 2017.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

ENS20. Muhammed F. Esgin, Ngoc Khanh Nguyen, and Gregor Seiler. Practical exact proofs from lattices: New techniques to exploit fully-splitting rings. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 259–288. Springer, Heidelberg, December 2020.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

FWK21. Valeh Farzaliyev, Jan Willemson, and Jaan Kristjan Kaasik. Improved lattice-based mix-nets for electronic voting. In *Information Security and Cryptology – ICISC 2021*. Springer International Publishing, 2021.

GHM$^+$21. Kristian Gjøsteen, Thomas Haines, Johannes Müller, Peter Rønne, and Tjerand Silde. Verifiable decryption in the head. Cryptology ePrint Archive, Report 2021/558, 2021.

Gjø11. Kristian Gjøsteen. The norwegian internet voting protocol. In Aggelos Kiayias and Helger Lipmaa, editors, *E-Voting and Identity - Third International Conference, VoteID 2011*, volume 7187 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

GM82. Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 365–377, New York, NY, USA, 1982. Association for Computing Machinery.

GMR85. S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

HMS21. Javier Herranz, Ramiro Martínez, and Manuel Sánchez. Shorter lattice-based zero-knowledge proofs for the correctness of a shuffle. Cryptology ePrint Archive, Report 2021/488, 2021.

HR16. Feng Hao and Peter Y. A. Ryan, editors. *Real-World Electronic Voting: Design, Analysis and Deployment.* CRC Press, 2016.

LN16. Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16*, volume 10052 of *LNCS*, pages 124–139. Springer, Heidelberg, November 2016.

LNP22. Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plancon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. Cryptology ePrint Archive, Report 2022/284, 2022. `https://ia.cr/2022/284`.

LNS21. Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 215–241. Springer, Heidelberg, May 2021.

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

LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, June 2015.

Nef01. C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 116–125. ACM Press, November 2001.

RST$^+$22. Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for spdz. *J. Cryptol.*, 35(1), jan 2022.

Scy. Scytl. Scytl sVote, complete verifiability security proof report - software version 2.1 - document 1.0.

Sil22. Tjerand Silde. Verifiable decryption for BGV. Workshop on Advances in Secure Electronic Voting, 2022. `https://ia.cr/2021/1693`.

Str19. Martin Strand. A verifiable shuffle for the GSW cryptosystem. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 165–180. Springer, Heidelberg, March 2019.

ZSS20. Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. COSAC: COmpact and scalable arbitrary-centered discrete gaussian sampling over integers. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 284–303. Springer, Heidelberg, 2020.

Appendix

A Verifiable Decryption Secure Against $n-1$ Parties

A.1 Optimistic Decryption

We present a new verifiable decryption protocol secure against $n-1$ parties. The protocol is inspired by the information theoretic MAC used in the MPC-protocols by Damgård *et al.* [DPSZ12,DKL$^+$13]. Here we start with an encrypted message m, sample a secret MAC α as the sum of independent shares α_j and check that the decryption of αm is correct. We use homomorphic encryption and standard commit-and-open techniques to ensure that the MAC value is honestly generated. Assuming at least one honest party, then the probability that the adversary can succeed is to guess the value of α before the decryption of αm is revealed. This happens with probability $1/|R_p| = 1/p^N$, which is negligible in the security parameter λ.

Note that this setting is usually not acceptable in an election, but it might still have value in at least the two following situations.

Firstly, as the protocol above is expensive both in computational complexity and memory consumption, it might be interesting to get a preliminary result of the election quickly by running the more lightweight protocol first and then heavier protocol afterwards. This way, we can within short time get a result with good confidence, and later get a proof that the first output was indeed correct.

Secondly, some smaller elections are organized such that each party voting is also computing a shuffle and a partial decryption, and hence, is both a voter and an infrastructure player in the election. In this case one has neither privacy nor integrity if everyone is colluding, and it makes sense to use a more lightweight protocol to compute the tally.

A.2 Homomorphic Multiplication

We describe additional algorithms to extend the BGV encryption protocol from Section 3.1 to include homomorphic multiplication of ciphertexts.

- `KeyGenExt`, runs `KeyGen` from Section 3 and outputs public key $\mathtt{pk} = (a, b) = (a, as + pe)$ and secret key $\mathtt{sk} = (s_1, s_2) = (s, s^2)$.

- `Mult`, on input two ciphertexts $\boldsymbol{c}_1 = (u_1, v_1)$ and $\boldsymbol{c}_2 = (u_2, v_2)$, computes $d_0 = v_1 \cdot v_2$, $d_1 = u_1 \cdot v_2 + u_2 \cdot v_1$ and $d_2 = u_1 \cdot u_2$, and outputs the product ciphertext $\boldsymbol{d} = (d_0, d_1, d_2)$.

- `DecExt`, takes as input a product ciphertext $\boldsymbol{d} = (d_0, d_1, d_2)$ and computes $m' = (d_0 - s_1 \cdot d_1 + s_2 \cdot d_2 \mod q) \mod p$. The algorithm outputs m'.

The output of the decryption algorithm is correct if $\|d_0 - s_1 \cdot d_1 + s_2 \cdot d_2\|_\infty = B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$. Furthermore, we present the extended passively secure distributed decryption technique used in the MPC-protocols by Damgård *et al.* [DPSZ12, DKL+13]. When decrypting, we assume that each decryption server $\mathcal{D}_j$, for $1 \leq j \leq \xi$, has a uniformly random share $\mathsf{sk}_j = (s_{1,j}, s_{2,j})$ of the secret key $\mathsf{sk} = (s_1, s_2)$ such that $s_1 = s_{1,1} + s_{1,2} + ... + s_{1,\xi}$ and $s_2 = s_{2,1} + s_{2,2} + ... + s_{2,\xi}$. Then they partially decrypt using the following algorithm:

- `DistDecExt`, on input a secret key-share sk_j and a ciphertext $\boldsymbol{d} = (d_0, d_1, d_2)$, computes $m_j = s_{1,j} \cdot d_1 - s_{2,j} \cdot d_2$, sample some large noise $E_j \leftarrow\!\!\$\ R_q$ such that $\|E_j\|_\infty \leq 2^{\mathsf{sec}}(B_{\mathtt{Dec}}/p\xi)$, and then outputs $t_j = m_j + pE_j$.

We obtain the full decryption of the ciphertext $\boldsymbol{d} = (d_0, d_1, d_2)$ as $m = (d_0 - t \mod q) \mod p$, where $t = t_1 + t_2 + ... + t_\xi$. This will give the correct decryption as long as the noise $\|d_0 - t\|_\infty \leq (1 + 2^{\mathsf{sec}})B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$.

A.3 Decryption Protocol

We use the same public parameters and ciphertexts as received in the decryption protocol in Section 5. Each decryption server has a secret key-share as described in `KeyGenExt`. Our verifiable decryption protocol works as following.

Partial decryption. Each party $\mathcal{D}_j$ run `DistDec`, as defined in Section 3.1, on each ciphertext $\boldsymbol{c}_i$ to produce partial decryptions $t_{i,j}$. Output $\{t_{i,j}\}_{i=1}^{\tau}$.

Commit to randomness. Each party samples a random $\alpha_j \leftarrow\!\!\$\ R_p$, run `Enc` with message α_j to produce ciphertext $\boldsymbol{c}_{\alpha_i} = (u_{\alpha_j}, v_{\alpha_j})$, and commit to the message and randomness as $h_j = \mathtt{H}(\alpha_j, r_{\alpha_j}, e_{\alpha_j}, e'_{\alpha_j})$, for a hash-function $\mathtt{H}$. Output h_j.

Output MAC shares. When each party $\mathcal{D}_j$ has published h_j, output $\boldsymbol{c}_{\alpha_j}$.

Compute MAC. Each party gather the encrypted MAC shares $\{\boldsymbol{c}_{\alpha_j}\}$ and sum them together as $\boldsymbol{c}_\alpha = \boldsymbol{c}_{\alpha_1} + ... + \boldsymbol{c}_{\alpha_\xi}$. Then, for each ciphertext $\boldsymbol{c}_i$, use `Mult` to compute the homomorphic multiplication with $\boldsymbol{c}_\alpha$ denoted $\boldsymbol{d}_i = (d_{i,0}, d_{i,1}, d_{i,2})$.

Commit to partial MAC decryption. Each party $\mathcal{D}_j$ runs the `DistDecExt` on each ciphertext $\boldsymbol{d}_i$ to produce partial decryptions $t_{\alpha_{i,j}}$. Commit to the partial decryption as $h_{\alpha_j} = \mathtt{H}(t_{\alpha_{1,j}}, ..., t_{\alpha_{\tau,j}})$. Output h_{α_j}.

Output partial MAC decryption. When each party $\mathcal{D}_j$ has published $h_{\alpha_{i,j}}$, output the partial MAC α_j, randomness $r_{\alpha_j}, e_{\alpha_j}, e'_{\alpha_j}$ and partial decryptions $\{t_{\alpha_{i,j}}\}_{i=1}^{\tau}$.

Verify correct decryption. A verifier computes $m_i = (v_{i,0} - t_i \mod q) \mod p$ for $t_i = t_{i,1} + ... + t_{i,\xi}$ and $m_{\alpha_i} = (d_{i,0} - t_{\alpha_i} \mod q) \mod p$ for $t_{\alpha_i} = t_{\alpha_{i,1}} + ... + t_{\alpha_{i,\xi}}$ for all $i = 1, ..., \tau$. Also compute $\alpha = \alpha_1 + ... + \alpha_\xi$. Then, check that $\boldsymbol{c}_{\alpha_i}$ was correctly computed, that h_j and $h_{\alpha_{i,j}}$ have valid openings and that $m_{\alpha_i} = m_i \cdot \alpha$ in R_p for all i. If all checks holds then output $\{m_i\}_{i=1}^{\tau}$ and otherwise output $\perp$.

A.4 Parameters and Size

Bounding the noise. We know from Section 7 that the amount of noise in each input ciphertext $\boldsymbol{c}_i$ is bounded by $B_{\boldsymbol{c}} = (\iota+1)p(2N+1)+\lceil(p-1)/2\rceil$. Furthermore, the noise in each ciphertext $\boldsymbol{c}_{\alpha_j}$ is bounded by $B_{\boldsymbol{c}_\alpha} = p(2N+1)+\lceil(p-1)/2\rceil$. It follows from Brakerski *et al.* [BGV12, Section 5.2] that the noise in each product ciphertext $\boldsymbol{d}_i$ is bounded by $B_{\boldsymbol{d}} = \sqrt{N}\cdot B_{\boldsymbol{c}}\cdot B_{\boldsymbol{c}_\alpha}$. Using `DistDecExt`, noise of size $2^{\mathsf{sec}}B_{\boldsymbol{d}}$ is added to $\boldsymbol{d}_i$. To ensure correct decryption, we must choose q such that $(1+2^{\mathsf{sec}})B_{\boldsymbol{d}} < q/2$. Inserting the parameters from Table 2 we get that $B_{\boldsymbol{d}} < 2^{38}$, and hence, we can choose $q \approx 2^{78}$ as in Section 7.

Total size. Each partial decryption consists of one ring-element, and each ring element can be represented with $N \log q$ bits. Each party $\mathcal{D}_j$ only sends two hashes of size 256 bits and the elements $\alpha_j, r_{\alpha_j}, e_{\alpha_j}, e'_{\alpha_j}$ which are of size $2N$ bits each to generate the MAC, which is essentially negligible compared to the ciphertexts and partial decryptions. The total size of the decryption protocol, not counting the input ciphertexts, is $\approx 2\xi\tau N\log q$. Using the parameters from above we get that the concrete size is $\approx 80\tau$ KB per decryption server.

B Security of the Voting Protocol

Here we provide a more formal description of the voting protocol described in Section 6, give security notions, sketch a security proof and discuss the security properties of the full voting protocol.

B.1 Verifiable Voting Schemes with Return Codes

A *verifiable cryptographic voting scheme* in our architecture is usually defined in terms of algorithms for the tasks of election setup, casting ballots, counting cast ballots and verifying the count. To support return codes, we also need algorithms for voter registration and pre-code computation. Finally, to accurately model the counting process, we need algorithms for shuffling and distributed decryption.

The *setup* algorithm `Setup` outputs a *public key* `pk`, *decryption key shares* $\mathtt{dk}_i$ and a *code key* `ck`.

The *register* algorithm `Reg` takes a public key `pk` as input and outputs a *voter verification key* `vvk`, a *voter casting key* `vck` and a function f from ballots to pre-codes.

The *cast* algorithm `Cast` takes a public key `pk`, a voter casting key `vck` and a *ballot* v, and outputs an *encrypted ballot* ev and a *ballot proof* π_v.

The *code* algorithm `Code` takes a code key `ck`, an encrypted ballot ev and a proof π_v as input and outputs a pre-code $\hat{r}$ or $\perp$. (If the code key `ck` is $\perp$, the algorithm outputs 0 or 1.)

The *shuffle* algorithm `Shuffle` takes a public key `pk` and a sequence of encrypted ballots $\boldsymbol{ev}$, and outputs a sequence of encrypted ballots $\boldsymbol{ev}'$ and a proof of shuffle π_s.

The *verify* algorithm $\texttt{Verify}$ takes a public key pk, two sequences of encrypted ballots $\boldsymbol{ev}$, and $\boldsymbol{ev}'$ and a proof π_s, and outputs 0 or 1.

The *distributed decryption* algorithm $\texttt{DistDec}$ takes a decryption key $\mathtt{dk}_i$ and a sequence of encrypted ballots $\boldsymbol{ev}$, and outputs a sequence of ballot decryption shares $\boldsymbol{sv}_i$ and a decryption proof $\pi_{d,i}$.

The *combining* algorithm $\texttt{Comb}$ takes a public key pk, a sequence of encrypted ballots $\boldsymbol{ev}$, ballot decryption share sequences $\boldsymbol{sv}_1, \boldsymbol{sv}_2, \dots, \boldsymbol{sv}_{l_d}$ with proofs $\pi_{d,1}, \pi_{d,2}, \dots, \pi_{d,l_d}$, and outputs either $\perp$ or a sequence of ballots $v_1, v_2, \dots, v_{l_t}$.

A cryptographic voting scheme is l_s*-correct* if for any $(\mathsf{pk}, \{\mathtt{dk}_i\}, \mathtt{ck})$ output by $\texttt{Setup}$ and any $(\mathtt{vvk}_1, \mathtt{vck}_1, f_1), \dots, (\mathtt{vvk}_{l_V}, \mathtt{vck}_{l_V}, f_{l_V})$ output by $\texttt{Reg}(\mathsf{pk})$, any ballots $v_1, \dots, v_{l_V}$, any $(ev_i^{(0)}, \pi_{v,i})$ output by $\texttt{Cast}(\mathsf{pk}, \mathtt{vck}_i, v_i)$, $i = 1, \dots, l_V$, any sequence of l_s sequences of encrypted ballots $\boldsymbol{ev}^{(j)}$ with proofs $\pi_{s,j}$ output by $\texttt{Shuffle}(\mathsf{pk}, \boldsymbol{ev}^{(j-1)})$, any ballot decryption shares $\boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}$ with proofs $\pi_{d,1}, \dots, \pi_{d,l_d}$ output by $\texttt{DistDec}(\mathtt{dk}_i, \boldsymbol{ev}^{(l_s)})$, $i = 1, 2, \dots, l_d$ and any $(v'_1, \dots, v'_{l_V})$ possibly output by $\texttt{Comb}(\mathsf{pk}, \boldsymbol{ev}^{(l_s)}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}), \pi_{d,1}, \dots, \pi_{d,l_d})$, then:

- $\texttt{Code}(\mathtt{ck}, \mathtt{vvk}_i, ev_i, \pi_{v,i}) = f_i(v_i)$, $\texttt{Code}(\perp, \mathtt{vvk}_i, ev_i, \pi_{v,i}) = 1$,
- $\texttt{Verify}(\mathsf{pk}, \boldsymbol{ev}^{(j-1)}, \boldsymbol{ev}^{(j)}, \pi_{s,j}) = 1$ for $j = 1, 2, \dots, l_s$,
- $\texttt{Comb}(\mathsf{pk}, \boldsymbol{ev}^{(l_s)}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}), \pi_{d,1}, \dots, \pi_{d,l_d})$ did not output $\perp$, and
- $v_1, \dots, v_{l_V}$ equals $v'_1, \dots, v'_{l_V}$, up to order.

We also require that the distribution of ev_i only depends on pk and v_i, not $\mathtt{vck}_i$.

For any such scheme we define a *decryption* algorithm $\texttt{Dec}$ that first applies a number of shuffles (possibly zero) to the single ciphertext, then applies $\texttt{DistDec}$ and $\texttt{Comb}$ in sequence. Note that this algorithm will not actually be used, but it simplifies the definition of security.

B.2 Our Scheme

Our voting scheme combines the BGV encryption together with our shuffle (Section 4) and distributed decryption (Section 5). We adapt the techniques from Aranha *et al.* [ABG+21] to get extractability and code voting, but omit the details.

- $\texttt{Setup}$ computes $\mathsf{pk}_C \leftarrow \texttt{KeyGen}_C$, $(\mathsf{pk}_V, \mathtt{dk}_V) \leftarrow \texttt{KeyGen}_{VE}$, $(\mathsf{pk}_R, \mathtt{dk}_R) \leftarrow \texttt{KeyGen}_{VE}$, as well as key shares $\mathtt{dk}_{V,i}$ for every decryption server. The public key $\mathsf{pk} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{pk}_R)$, the decryption share is $\mathtt{dk} = (\mathsf{pk}_C, \mathtt{dk}_{V,i})$ and the code key is $\mathtt{ck} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathtt{dk}_R)$.
- $\texttt{Reg}$ takes $\mathsf{pk} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{pk}_R)$ as input. It samples $a \leftarrow\$ R_p$ and computes $(\boldsymbol{c}_a, d_a) \leftarrow \texttt{Com}(\mathsf{pk}_C, a)$. The voter verification key is $\mathtt{vvk} = \boldsymbol{c}_a$, the voter casting key is $(a, \boldsymbol{c}_a, d_a)$, and the function f is $v \mapsto v + a$.
- $\texttt{Cast}$ takes $\mathsf{pk} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathsf{pk}_R)$, $\mathtt{vck} = (a, \boldsymbol{c}_a, d_a)$ and v as input. It computes $\boldsymbol{v} \leftarrow \texttt{Enc}_{VE}(\mathsf{pk}_V, v)$, $\hat{r} \leftarrow a + v$ and $\boldsymbol{w} \leftarrow \texttt{Enc}_{VE}(\mathsf{pk}_R, \hat{r})$, along with a

proof $\pi_{v,0}$ that $\boldsymbol{v}$ and $\boldsymbol{w}$ are well-formed ciphertexts, and that $\boldsymbol{c}_a$ is a commitment to the difference of the decryptions. The encrypted ballot is $ev = \boldsymbol{v}$, while the ballot proof is $\pi_v = (\boldsymbol{w}, \pi_{v,0})$.

- $\mathtt{Code}$ takes $\mathtt{ck} = (\mathsf{pk}_C, \mathsf{pk}_V, \mathtt{dk}_R)$, a voter verification key $\mathtt{vvk}$, an encrypted ballot $ev = \boldsymbol{v}$ and a ballot proof $\pi_v = (\boldsymbol{w}, \pi_{v,0})$ as input. It verifies $\pi_{v,0}$ and outputs $\perp$ if verification fails. Otherwise, it computes $\hat{r} \leftarrow \mathtt{Dec}_{VE}(\mathtt{dk}_R, \boldsymbol{w})$ and outputs $\hat{r}$. (If $\mathtt{ck} = \perp$, it outputs 1 if and only if it accepts $\pi_{v,0}$.)
- The *shuffle* algorithm $\mathtt{Shuffle}$ and the *verify* algorithm $\mathtt{Verify}$ are as described in Section 4. The *distributed decryption* algorithm $\mathtt{DistDec}$ and the *combining* algorithm $\mathtt{Comb}$ are as described in Section 5.

It is straight-forward to verify that the scheme is correct.

B.3 Security Notions

Our notion of confidentiality is similar to the usual ballot box privacy notions [BCG+15]. An adversary that sees both the contents of the ballot box, the intermediate shuffles and the decrypted ballot shares should not be able to determine who cast which ballot. This should hold even if the adversary can see pre-codes, learn the code key, some voter casting keys and some decryption key shares, insert adversarially generated ciphertexts into the ballot box, introduce adversarially generated intermediate shuffles and publish adversarially chosen decrypted ballot shares.

Our notion of integrity is again fairly standard, adapted to return codes. An adversary should not be able to cause an incorrect pre-code or inconsistent decryption or non-unique decryption, even if the adversary knows all of the key material.

We define security notions for a verifiable cryptographic voting scheme using an experiment where an adversary $\mathcal{A}$ is allowed to reveal keys, make challenge queries, create ciphertexts, ask for ciphertexts to be shuffled, create shuffles, and ask for ballot shares. We use this experiment to define games both for confidentiality and for integrity. The experiment works as follows:

- Sample $b, \leftarrow\!\!\$ \{0,1\}$. Set L, L', L'' to be empty lists.
- $(\mathsf{pk}, \{\mathtt{dk}_i\}, \mathtt{ck}) \leftarrow \mathtt{Setup}$. For $i = 1, \ldots, l_V$: $(\mathtt{vvk}_i, \mathtt{vck}_i, f_i) \leftarrow \mathtt{Reg}(\mathsf{pk})$. Send $(\mathsf{pk}, \mathtt{vvk}_1, \ldots, \mathtt{vvk}_{l_V})$ to $\mathcal{A}$.
- On a *voter reveal query* i, send $(\mathtt{vck}_i, f_i)$ to $\mathcal{A}$. On a *decrypt reveal query* i, send $\mathtt{dk}_i$ to $\mathcal{A}$. On a *code reveal query*, send $\mathtt{ck}$ to $\mathcal{A}$.
- On a *challenge query* (i, v_0, v_1), compute $(ev, \pi_v) \leftarrow \mathtt{Cast}(\mathsf{pk}, \mathtt{vck}_i, v_b)$, $\hat{r} \leftarrow \mathtt{Code}(\mathtt{ck}, \mathtt{vvk}_i, ev, \pi_v)$, append (i, v_0, v_1, ev, π_v) to L. Send (ev, π_v) to $\mathcal{A}$.
- On a *chosen ciphertext query* (i, ev, π_v), compute $\hat{r} \leftarrow \mathtt{Code}(\mathtt{ck}, \mathtt{vvk}_i, ev, \pi_v)$. If $\hat{r} \neq \perp$, append $(i, \perp, \perp, ev, \pi_v)$ to L. Send $\hat{r}$ to $\mathcal{A}$.
- On a *shuffle query* $\boldsymbol{ev}$, compute $(\boldsymbol{ev}', \pi_s) \leftarrow \mathtt{Shuffle}(\mathsf{pk}, \boldsymbol{ev})$, then record $(\boldsymbol{ev}, \boldsymbol{ev}', \pi_s)$ in L'. Send $(\boldsymbol{ev}', \pi_s)$ to $\mathcal{A}$.
- On a *chosen shuffle query* $(\boldsymbol{ev}, \boldsymbol{ev}', \pi_s)$, we record it in L' if and only if $\mathtt{Verify}(\mathsf{pk}, \boldsymbol{ev}, \boldsymbol{ev}', \pi_s) = 1$.

– On a *ballot decryption share query* $(i, \boldsymbol{ev})$, we then compute $(\boldsymbol{sv}_i, \pi_{d,i}) \leftarrow$ $\texttt{DistDec}(\texttt{dk}_i, \boldsymbol{ev})$, record $(i, \boldsymbol{ev}, \boldsymbol{sv}_i, \pi_{d,i})$ in L'' and send $(\boldsymbol{sv}_i, \pi_{d,i})$ to $\mathcal{A}$.
– On a *test query* $(\boldsymbol{ev}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}, \pi_{d,1}, \dots, \pi_{d,l_d})$, then compute the *result* $\leftarrow$ $\texttt{Comb}(\texttt{pk}, \boldsymbol{ev}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}, \pi_{d,1}, \dots, \pi_{d,l_d})$ and send *result* to $\mathcal{A}$.

Eventually, the adversary outputs a bit b'.

The confidentiality game follows the usual left-or-right game pattern, where an adversary makes challenge queries and must determine the value of the bit b. The test query is irrelevant for the confidentiality game.

The integrity game follows the usual pattern where the adversary's goal is to achieve certain inconsistencies, either during a code query or during a test query. The inconsistencies are that a pre-code does not match the encrypted ballot, that an outcome verifies as correct but is inconsistent with the challenge ciphertexts chosen for counting, or that there is no unique decryption. (The test query is not strictly needed. We could have had the adversary output its encrypted ballots and ballot decryption shares instead of making a test query. But the test query pattern is convenient in many similar settings, so we include it.) The bits b, b' are not really used in the game for integrity, nor is the shuffle query. The challenge query is used to create honestly encrypted ballots.

Confidentiality fails trivially if the counting phase trivially reveals the challenge bit. This happens unless the left hand ballots and the right-hand ballots are identical, up to order. (Recall that the adversary should figure out who cast which ballots, not what ballots were cast.) Confidentiality also fails trivially if the adversary makes more than one challenge query or chosen ciphertext query for any given voter. And confidentiality fails trivially if the adversary reveals too much key material. We should not count executions where confidentiality fails trivially towards the adversary's advantage. Technically, we count this using a *freshness* event when evaluating the advantage.

In an execution of this experiment, we say that a sequence of encrypted ballots $\boldsymbol{ev}$ is *valid* if tuples $(i_1, v_{01}, v_{11}, ev_1, \pi_{v,1}), \dots, (i_{l_c}, v_{0l_c}, v_{1l_c}, ev_{l_c}, \pi_{v,l_c})$ in L and L' contains a sequence of tuples $(\boldsymbol{ev}^{(j-1)}, \boldsymbol{ev}^{(j)}, \pi_{s,j})$, $j = 1, 2, \dots, l_s$, such that $\boldsymbol{ev}^{(0)} = (ev_1, \dots, ev_{l_c})$ and $\boldsymbol{ev}^{(l_s)} = \boldsymbol{ev}$. In this case we also say that $\boldsymbol{ev}$ *derives* from $(i_1, v_{01}, v_{11}, ev_1, \pi_{v,1}), \dots, (i_{l_c}, v_{0l_c}, v_{1l_c}, ev_{l_c}, \pi_{v,l_c})$. A valid sequence $\boldsymbol{ev}$ is *honest* if at least one of the tuples $(\boldsymbol{ev}^{(j-1)}, \boldsymbol{ev}^{(j)}, \pi_{s,j})$ originated with a shuffle query. A valid sequence $\boldsymbol{ev}$ is *balanced* if the ballot sequence $(v_{01}, \dots, v_{0l_c})$ equals $(v_{11}, \dots, v_{1l_c})$, up to order.

We define events related to confidentiality and integrity. Let E_g be the event that $b = b'$. Let E_f denote the event that an execution is *fresh*, which is true if the following are satisfied: there is no decrypt reveal query for at least one i; for any i, there is either no challenge query, or at most one challenge query and no voter reveal query or chosen ciphertext query; and for any ballot decryption share query $(\cdot, \boldsymbol{ev})$, the sequence $\boldsymbol{ev}$ is balanced and honest *at the time of the ballot decryption share query.*

Let F_i (incorrect pre-code) be the event that for some chosen ciphertext query (i, ev, π_v) where $\texttt{Code}(\texttt{ck}, \texttt{vvk}_i, ev, \pi_v) = \hat{r} \neq \perp$, we have that either $\texttt{Dec}(\{\texttt{dk}_i\}, ev) = \perp$ or $\texttt{Dec}(\{\texttt{dk}_i\}, ev) = v$ and $f_i(v) \neq \hat{r}$.

Let F_c (count failure) be the event that a test query gets *result* $= \perp$ when $\boldsymbol{ev}$ is valid and $(\boldsymbol{ev}, \boldsymbol{sv}_i, \pi_{d,i})$ is in L'' for $i = 1, \dots, l_d$.

Let F_d (inconsistent decryption) be the event that a test query $(\boldsymbol{ev}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}, \pi_{d,1}, \dots, \pi_{d,l_d})$ with *result* $= (v_1, \dots, v_{l_c})$, where $\boldsymbol{ev}$ derives from $(i_1, v_{01}, v_{11}, ev_1, \pi_{v,1}), \dots, (i_{l_c}, v_{0l_c}, v_{1l_c}, ev_{l_c}, \pi_{v,l_c})$, there is no permutation π on $\{1, 2, \dots, l_c\}$ such that $v_{b,k} = \perp$ or $v_{b,k} = v_{\pi(k)}$ for $k = 1, 2, \dots, l_c$. Let F_u (no unique decryption) be the event that two test queries $(\boldsymbol{ev}, \boldsymbol{sv}_1, \dots, \boldsymbol{sv}_{l_d}, \pi_{d,1}, \dots, \pi_{d,l_d})$ and $(\boldsymbol{ev}, \boldsymbol{sv}'_1, \dots, \boldsymbol{sv}'_{l_d}, \pi_{d,1}', \dots, \pi_{d,l_d}')$ for some valid $\boldsymbol{ev}$ get results *result* and *result*$'$ that are not equal up to order, and neither of which are equal to $\perp$. The advantage of the adversary is

$$\max\{2 \cdot |\Pr[E_g \wedge E_f] - \Pr[E_f]/2|, \Pr[F_i \vee F_c \vee F_d \vee F_u]\}.$$

B.4 Security Proof Sketch

We briefly sketch a proof for how to bound the advantage of an adversary against the cryptographic voting scheme in terms of adversaries against the shuffle, the distributed decryption scheme, the commitment scheme or the encryption scheme.

Confidentiality. We begin by analyzing the confidentiality event $\Pr[E_g \wedge E_f]$.

The proof would proceed as a sequence of games, where the first game is the interaction between the experiment and the adversary.

In the next game, we stop the adversary with a forced guess $b' = 0$ immediately upon any query that would make the execution non-fresh. Note that a query that makes the execution non-fresh can be recognized with no secret information, and at the time the query is made. A brief computation shows that this changes nothing, but in the further analysis we may assume that the execution remains fresh.

We next simulate all the zero knowledge proofs involved, which is straightforward in the random oracle model since all our proofs are HVZK.

Next, we change the challenge query so that instead of computing the precode as $\hat{r} = a + v$, it samples $\hat{r}$ uniformly at random. If this change is observable, we get an adversary against hiding for the commitment scheme.

Next, for any ballot decryption share query for a sequence $(ev_1, \dots, ev_{l_c})$, we decrypt ev_i to v_i, then use the HVZK simulator from Section 5 to simulate the decryption share given the decryption v_i. This change is unobservable. (To get an adversary, we guess a decryption key share i for which the adversary will never make a decrypt reveal query, and simulate the other decryption key shares as random shares. When the adversary makes a ballot decryption share query for i, we compute the ballot decryption shares for the other decryption key shares and compute the ith ballot decryption share to give the correct result when combined.)

Next, for chosen ciphertext queries, we decrypt the $\boldsymbol{w}$ using $\mathtt{dk}_R$ and subtract a to recover v, and then record (i, v, v, ev, π_v) instead of $(i, \perp, \perp, ev, \pi_v)$. By the soundness of the ballot proof (details omitted), we now have that every tuple (i, v_0, v_1, ev, π_v) in L satisfies $\texttt{Dec}(\texttt{dk}_V, ev) = v_b$.

Next, for any ballot decryption share query for an honest and balanced sequence $(ev_1, \dots, ev_{l_c})$ deriving from $(\cdot, v_{01}, v_{11}, \cdot, \cdot), \dots, (\cdot, v_{0l_c}, v_{1l_c}, \cdot, \cdot)$, sample a permutation π on $\{1, 2, \dots, l_c\}$ and use $v_i = v_{0\pi(i)}$ instead of decrypting ev_i. If this change is observable, we either get an adversary against soundness for the shuffle (when the decryption of the output of a shuffle is not equal to the decryption of the input to the shuffle, up to order) or an adversary against the encryption scheme (when the adversary notices that the ballot decryption shares are inconsistent with the encrypted ballots). The latter adversary re-randomizes the shuffle with random values instead of encryptions of zero.

At this point, the decryption key shares $\{\texttt{dk}_i\}$ are no longer used. Also, the pre-code encrypted in the challenge query is independent of the challenge ballots.

Finally, for challenge queries we encrypt a random ballot instead of the left or right ballot. If this change is observable, we get a real-or-random adversary against the encryption scheme.

At this point, the challenge bit b is no longer used. It follows that the adversary has no advantage in this game. By the above arguments, the claim that the difference between $\Pr[E_g \wedge E_f]$ and $\Pr[E_f]/2$ is appropriately bounded follows.

Integrity. Next, we analyze the integrity events. In this case, the adversary may have revealed every secret key, and there is no need for the execution to be fresh.

If a chosen ciphertext query results in an incorrect pre-code, then we immediately get an adversary against the soundness of the ballot proof (details omitted). It follows that the probability of F_i happening is appropriately bounded.

In the event that F_c happens, note that every encrypted ballot either originates with a challenge query or a chosen ciphertext query, the shuffles applied to the encrypted ballots originate with shuffle queries or chosen shuffle queries, and the ballot decryption shares all originate with ballot decryption share queries. By the completeness and soundness of the various arguments, and the bound on the number of shuffles, we get that the probability of F_c happening is appropriately bounded.

In the event that F_d happens, then either the output of some shuffle does not decrypt to the same as the input to the shuffle, in which case we get an adversary against the soundness of the shuffle, or the distributed decryption does not decrypt correctly, in which case we get an adversary against the soundness of the distributed decryption. It follows that the probability of F_d happening is appropriately bounded.

In the event that F_u happens, then either the decryption of the encrypted ballots is not unique, in which case we get an adversary against the soundness of the proofs ensuring valid ciphertexts (in the ballot proofs and the shuffle proofs), or one or both results are incorrect, in which case we get an adversary against the soundness of the shuffle. It follows that the probability of F_u happening is appropriately bounded.

The claim that $\Pr[F_i \vee F_c \vee F_d \vee F_u]$ is appropriately bounded follows.

B.5 Voting System Security Properties

Integrity. Integrity for a voting system is modeled using a game between an adversary and a set of voters, some of which may be corrupt. The adversary tells the honest voters what ballots to cast. If the count phase eventually runs and ends with a result, the adversary wins if the result is inconsistent with the ballots accepted as cast by the honest voters. (Recall that only the voter's last ballot cast is counted, so if the voter first accepts a ballot as cast, and then tries to cast another ballot and this fails, the end result is that they have not accepted a ballot as cast.)

We can define a variant notion called ϵ-integrity where we allow a small error, and say that the adversary wins if the result is inconsistent with any $(1 - \epsilon)$ fraction of the ballots accepted as cast by the honest voters. (We need this since return codes for a single voter must be human-comparable, and can therefore collide with some non-negligible probability.)

Analysis. The voter will only accept the ballot as cast if the correct return code is received. If the correct return code is received, then the correct pre-code must have been computed at some point (except with some small probability due to collisions in the PRF).

If the *return code generator* $\mathcal{R}$ is honest, integrity of the cryptographic voting scheme implies that this can only happen if the correct ballot has been encrypted. If the auditor $\mathcal{A}$ is honest, the result will only be accepted if the encrypted ballot has been included among those sent to the first shuffler. By the integrity of the cryptographic voting system, all such ballots must then be included in the result.

If the voter's computer P and the ballot box $\mathcal{B}$ and the auditor $\mathcal{A}$ are honest, the the encrypted ballot will be included among those sent to the first shuffler. By the integrity of the cryptographic voting system, all such ballots must then be included in the result.

If a voter receives a return code without casting a ballot, the voter will no longer accept their ballot as cast.

In summary, ϵ-integrity holds if the auditor and either both the voters' phones and the return code generator are honest, or both the voters' computers and the ballot box are honest.

Verifiability. In a verifiable voting protocol, every voter gets a *receipt* after accepting a ballot as cast. Also, the auditor outputs a result and a *transcript.* Also, there is an algorithm for verifying either a transcript, a result and optionally a receipt.

Consider an execution of the voting protocol where the auditor outputs a result and a transcript. Then there is a set of honest voters with honest computers that accept their ballot as cast with some receipt, and for which the verification algorithm accepts the transcript, the result and their receipt. We say that a system is *verifiable* if the result is consistent with the list of these voters' ballots being included in the result.

Note that verifiability in and of itself does not guarantee anything about the correctness of the result. Instead, verifiability is best thought of as a tool that can be used to achieve trust in election integrity under fairly weak trust assumptions. For instance, one can prove that if a sufficiently large and hard to guess subset of voters run the verification algorithm on the transcript, result and their receipt, then the overall election has ϵ-integrity for some ϵ. (The "hard to guess" part is instrumental in proving this result. If the set of voters verifying an election is not hard to guess, achieving election integrity is much more difficult, at least without strong trust assumptions.)

Note also that we assume that the honest voter's computer is honest in the definition. If the voter's computer is corrupted, we are left with considering integrity as above, which can be achieved conditional on other players being honest.

Analysis. Verifiability for our protocol follows by integrity for the underlying cryptosystem, since the execution of our protocol can be thought of as an interaction with the experiment for the underlying cryptosystem, where the honest computer's actions correspond to challenge queries, and part of the verification algorithms' work correspond to a test query. The structure of the protocol then ensures that if the result output by a test query is inconsistent with corresponding ballots output by challenge queries, integrity fails for the underlying cryptosystem.

In summary, if the underlying cryptosystem has integrity, the voting protocol is verifiable.

Privacy. Privacy for a voting protocol is modeled as a left-or-right game with an adversary and a set of voters, some of which may be corrupt. The adversary gives pairs of ballots to honest voters, and they will all either cast the left ballot or the right ballot. The adversary must decide which they cast. (This essentially amounts to deciding who cast which ballot.)

The adversary can corrupt players and also control the network. We shall assume that players use secure channels to communicate. This means that only the fact that players are communicating and the length of their communications leak. Since message flows and message lengths are fixed and public knowledge, we can ignore the network in the subsequent analysis.

We want to avoid adversaries that deduce the honest voters' ballots trivially from the result, so we require that the adversary organizes the pairs of ballots given to the honest voters in such a way that the ballots cast by the honest voters are independent of whether the voters cast the left or the right ballot.

Analysis. If some honest voter's *computer P* is compromised, the adversary can trivially win the privacy game.

If every *shuffle server* is compromised, the adversary learns the correspondence between decrypted ballots and voters, and can trivially win the privacy game.

If every *decryption server* is compromised, the adversary learns the decryption key, and can trivially win the privacy game.

If a voter casts more than one ballot, a compromised *return code generator* or *voter phone* will always be able to decide if they are the same or not by observing the return code sent to the voter. If the ballots are distinct, the return code generator will learn information about which ballots were submitted, and typically learn both ballots. (We could prove privacy when the voter casts more than one ballot and the return code generator and the voter phone are both honest, but this requires adding a restricted challenge query that does not reveal the precode to the cryptosystem experiment.)

Suppose the honest voters cast at most one ballot each, their computers remain honest, and at least one shuffle server and one decryption server is honest. Then privacy follows from confidentiality of the cryptographic voting system, since the protocol execution can be interpreted as an interaction with the cryptosystem experiment and the protocol together with our assumptions ensure a fresh execution.

Note that cut-and-paste attacks against confidentiality, which commonly affect this type of voting protocol, do not work against this protocol because the ballot proof includes an encryption of the return code and a proof that the return code is correct which is tied to the voter's public key material. Cut-and-paste attacks would anyway constitute a valid attack on the cryptosystem.

In summary, privacy holds if the honest voters' computers are honest, there is at least one honest shuffle server and one honest decryption server, and no honest voter casts more than one ballot.

C Proofs for Amortized Exact ZKPoPK

C.1 Proof of Lemma 1.

Proof. First, we construct an extractor $\mathcal{E}$ as in [BLNS21] which does the following 8 times:

1. First, run $\mathtt{P}^*$ on random challenges from $\mathtt{V}$ until an accepting transcript is found. Abort if none is found after $8/\epsilon$ steps.
2. Let I_1 be the challenge set where $\mathtt{P}^*$ responded and let $\boldsymbol{E}|_{I_1}, \mathtt{MerklePaths}_{I_1}$ be the opened columns and Merkle tree paths. Fixing the other challenges, $\mathcal{E}$ adaptively reruns the proof for different challenges $I_2, I_3, \dots$ that contain so far unopened columns and collects these. If any collision in the Merkle tree is found then $\mathcal{E}$ outputs the hash collision and terminates, otherwise it continues until it collected a set J of at least k columns, or until $8\frac{k-\eta}{\epsilon/2-(k/(l-\eta))^\eta}$ time passed.
3. Finally, $\mathcal{E}$ re-runs $\mathtt{P}^*$ $16/\epsilon$ times with completely fresh challenges, obtaining new $\boldsymbol{E}|_I, \mathtt{MerklePaths}_I$. If for some of these instances the accepting transcript contains a hash collision in the Merkle tree (colliding with J) or $c_{\boldsymbol{d}}$ is opened with a different opening than in the first step, then output the hash collision or the respective two different openings of $c_{\boldsymbol{d}}$.

Using a standard heavy-row argument, [BLNS21] show that $\mathcal{E}$'s total runtime is bounded by the term mentioned in the Lemma. Moreover, define S as the event that $\mathtt{P}^*$ outputs a valid proof in the last step and C be the event that the values $\mathtt{P}^*$ outputs to $\mathcal{E}$ in the last step of the extractor are consistent (i.e. no hash collisions and the commitment was not opened differently than before). Then [BLNS21] show that it must hold that $\Pr[S \wedge C] > \epsilon/2$. We now show that if we cannot use J to decode to a valid witness, then the success probability of $\mathtt{P}^*$ must be lower than the given bound.

Define C' to be the RS code obtained from C (generated by `Encode`) when restricted to the indices of J. As $|J| = k$, C' has length k and minimum distance $d' = k - k' + 1$. For any $\boldsymbol{x} \in \mathbb{Z}_q^k$ we define the minimum distance of $\boldsymbol{x}$ to C' as $d'(C', \boldsymbol{x}) = \min_{\boldsymbol{c} \in C'} d(\boldsymbol{c}, \boldsymbol{x})$.

Let $\boldsymbol{E}^* := \boldsymbol{E}|_J$ be the matrix that was extracted by the extractor and let $\boldsymbol{d}^*$ be the opening message of the commitment. Assume that there exists $\boldsymbol{x} \in \mathbb{Z}_q^{3\tau+4}$ such that $d'(C', \boldsymbol{x}\boldsymbol{E}^*) \geq d'/3$. Then by [BCG+17, Appendix B], any random linear combination of $\boldsymbol{E}^*$ (in particular, we compute and output such a combination in the proof as $\overline{\boldsymbol{h}}$) has distance $\geq d'/6$ from C' except with probability $1/(q-\tau)$. Similar as in [BLNS21] we can use this to deduce that in such a case, it must hold that

$$\epsilon/2 < \frac{1}{q-\tau} + \left(1 - \frac{k-k'}{6l}\right)^{\eta}$$

which contradicts the bound on ϵ in this lemma. Therefore, each row of $\boldsymbol{E}^*$ must be within $d'/3$ of C', meaning that it is efficiently decodable. Let $\boldsymbol{h}^*, \boldsymbol{s}_0^*, \boldsymbol{s}_{i,j}^*, \boldsymbol{v}_0^*, \boldsymbol{v}_{i,j}^*$ be the respective decoded values and $\boldsymbol{r}_h^*, \boldsymbol{r}_0^*, \boldsymbol{r}_{i,j}*$ be the randomness. We consider the composition of the aforementioned row values as $\boldsymbol{V}$ and the randomness used in the encoding as $\boldsymbol{R}$, By applying another result from [BCG+17, Appendix B] we have that for any vector $\boldsymbol{y} \in \mathbb{Z}_q^{3\tau+4}$ it holds that $d'(\mathtt{Encode}_J(\boldsymbol{y}\boldsymbol{V}, \boldsymbol{y}\boldsymbol{R}), \boldsymbol{y}\boldsymbol{E}^*) < d'/3$. In other words, any linear transformation $\boldsymbol{y}$ when applied to the possibly noisy codewords $\boldsymbol{E}^*$ is within distance $d'/3$ of the codeword obtained from encoding $\boldsymbol{V}, \boldsymbol{R}$ after applying the same transformation $\boldsymbol{y}$. That means that $\overline{\boldsymbol{f}}$ is constructed from $\boldsymbol{s}_0^*, \boldsymbol{s}_{i,j}^*$ as we would expect.

Similar as in [BLNS21, Corollary 3.7] one can show that if there are $\leq (\epsilon/4)(q-\tau)$ choices of x such that

$$\overline{\boldsymbol{f}} = \ell_0(x)\boldsymbol{s}_0^* + \sum_{i=1}^{\tau}\sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{s}_{i,j}^* \tag{3}$$

$$\frac{1}{\ell_0(x)} \cdot \bar{\boldsymbol{f}} \circ \left[\bar{\boldsymbol{f}} - \boldsymbol{1}\right] \circ \left[\bar{\boldsymbol{f}} + \boldsymbol{1}\right] = \ell_0(x)\boldsymbol{v}_0^* + \sum_{i=1}^{\tau}\sum_{j=0}^{2} \ell_i(x)\ell_0(x)^j \boldsymbol{v}_{i,j}^* \tag{4}$$

then $\epsilon < 4(1 - \frac{2(k-k')}{3l})^\eta$, contradicting the bound on ϵ in the lemma. By multiplying 4 with $\ell_0(X)$ we obtain the equation

$$\bar{\boldsymbol{f}} \circ (\bar{\boldsymbol{f}} - \mathbf{1}) \circ (\bar{\boldsymbol{f}} + \mathbf{1}) - \ell_0(X)^2 \boldsymbol{v}_0^* - \sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(X) \ell_0(X)^{j+1} \boldsymbol{v}_{i,j}^* = \mathbf{0}. \qquad (5)$$

Replacing $\bar{\boldsymbol{f}}$ according to Equation 3 means that the above expression is of degree at most $9 \cdot \tau$. But it is 0 for more choices of x because $\epsilon > 36\tau/(q-\tau)$, meaning that the expression itself must be the zero-polynomial. Reducing Equation 5 modulo $\ell_0(X)$ and knowing that all $\ell_i(X)$ are independent, it follows that for all $i \in [\tau]$ the value $\boldsymbol{s}_{i,0}^*$ is in $\{-1, 0, 1\}^{vN}$.

Additionally, we have that

$$\boldsymbol{d}^* = \frac{1}{\ell_0(X)} \cdot \left(\sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(X) - \boldsymbol{A}\bar{\boldsymbol{f}} \right)$$

and replacing again $\bar{\boldsymbol{f}}$ with Equation 3 yields

$$\boldsymbol{d}^* = -\boldsymbol{A}\boldsymbol{s}_0^* + \frac{1}{\ell_0(X)} \cdot \left(\sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(X) - \boldsymbol{A} \left[\sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x) \ell_0(x)^j \boldsymbol{s}_{i,j}^* \right] \right)$$

Since $\boldsymbol{d}^*$ has been committed to before x is chosen, it must be that $\boldsymbol{d}^*$ is the constant of the polynomial on the right, so we have that $\boldsymbol{d}^* = -\boldsymbol{A}\boldsymbol{s}_0^*$ and therefore

$$\sum_{i=1}^{\tau} \sum_{j=0}^{2} \ell_i(x) \ell_0(x)^j \boldsymbol{A}\boldsymbol{s}_{i,j}^* = \sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(X).$$

Again reducing modulo $\ell_0(X)$ reveals that

$$\sum_{i=1}^{\tau} \ell_i(x) \boldsymbol{A}\boldsymbol{s}_{i,0}^* = \sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(X)$$

and by the independence of the $\ell_i(X)$ modulo $\ell_0(X)$ we have that $\boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_{i,0}^*$ for all $i \in [\tau]$, which proves the claim. □

C.2 Zero-Knowledge

Lemma 3. *There exists an efficient simulator $\mathcal{S}$ which, given $x, \beta_0, \beta_{1,0}, \ldots, \beta_{\tau,2}, I$ outputs a protocol transcript of the the protocol in Figure 3 whose distribution is indistinguishable from a real transcript between an honest prover and honest verifier.*

Proving Honest-Verifier Zero-Knowledge is sufficient for our application, as we will use the Fiat-Shamir transform to generate the challenges in Π_{AEx}.

Proof. Towards constructing a simulation, observe that

1. $\mathcal{M}$ and its opened paths do not reveal any information about the unopened columns as the commitment scheme used in creating $\mathcal{M}$ is hiding.
2. $\bar{\boldsymbol{f}}, \bar{\boldsymbol{r}}_f, \bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h$ are uniformly random due to the uniform choice of $\boldsymbol{s}_0, \boldsymbol{r}_0, \boldsymbol{h}, \boldsymbol{r}_h$.
3. Each encoded row of $\boldsymbol{E}$ uses η bits of randomness, so revealing η columns does not leak any information about the message being committed in the respective row.

Thus, for the proof we let $\mathcal{S}$ choose uniformly random $\bar{\boldsymbol{f}}, \bar{\boldsymbol{r}}_f, \bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h$. This allows the prover also to compute $\boldsymbol{d}$ consistently as $\frac{1}{\ell_0(x)} \cdot (\sum_{i=1}^{\tau} \boldsymbol{t}_i \ell_i(x) - \boldsymbol{A}\bar{\boldsymbol{f}})$, which has the same uniform distribution as in the real protocol, and thereby fix $c_{\boldsymbol{d}}$. Next, we let $\mathcal{S}$ choose all but the first two rows of $\boldsymbol{E}|_I$ uniformly at random. The second row will be computed according to x thus fulfilling the check on the encoding of $\bar{\boldsymbol{f}}, \bar{\boldsymbol{r}}_f$, while the first row is computed according to $\beta_0, \beta_{i,j}$ for the encoding of $\bar{\boldsymbol{h}}, \bar{\boldsymbol{r}}_h$. $\mathcal{S}$ now fixes the remaining columns of $\boldsymbol{E}$ as $\boldsymbol{0}$ and commits honestly to these as in the protocol. $\square$

Paper IV

Verifiable Decryption in the Head

Kristian Gjøsteen, Thomas Haines, Johannes Muller, Peter Rønne and Tjerand Silde

Accepted at the Australasian Conference on Information Security and Privacy, ACIPS 2022. The full version is available at: eprint.iacr.org/2021/558.pdf.

Verifiable Decryption in the Head

Kristian Gjøsteen[1], Thomas Haines[1,2], Johannes Müller[3],
Peter Rønne[3,4], and Tjerand Silde[1]

[1] Norwegian University of Science and Technology
{kristian.gjosteen,tjerand.silde}@ntnu.no
[2] Australian National University
thomas.haines@anu.edu.au
[3] University of Luxembourg
johannes.mueller@uni.lu
[4] Université de Lorraine, CNRS, LORIA
peter.roenne@gmail.com

Abstract. In this work we present a new approach to verifiable decryption which converts a 2-party passively secure distributed decryption protocol into a 1-party proof of correct decryption. To introduce our idea, we present a toy example for an ElGamal distributed decryption protocol that we also give a machine checked proof of, in addition to applying our method to lattices. This leads to an efficient and simple verifiable decryption scheme for lattice-based cryptography, especially for large sets of ciphertexts; it has small size and lightweight computations as we reduce the need of zero-knowledge proofs for each ciphertext. We believe the flexibility of the general technique is interesting and provides attractive trade-offs between complexity and security, in particular for the interactive variant with smaller soundness.
Finally, the protocol requires only very simple operations, making it easy to correctly and securely implement in practice. We suggest concrete parameters for our protocol and give a proof of concept implementation, showing that it is highly practical.

Keywords: verifiable decryption · distributed decryption · lattice-based crypto · MPC-in-the-Head · zero-knowledge proof · implementation

1 Introduction

There are many applications where we not only need to decrypt a ciphertext, but also prove that we have decrypted the ciphertext correctly without revealing the secret key. This is called *verifiable decryption*. Examples include mix-nets used for anonymous communication [SSA+18], decryption of ballots in electronic voting [HM20], and various uses of verifiable fully homomorphic encryption [LW18]. In particular, such applications usually require the decryption of a large number of ciphertexts.

It is well-known how to do verifiable decryption for public-key encryption schemes based on discrete logarithms (for ElGamal, proving the equality of two discrete logarithms [CP92] will do). Except for the recent publication by Lyubashevsky *et al.* [LNS21] (which provides a rather complicated decryption proof by combining proofs of linear relations, multiplications and range proofs), no efficient and straight-forward zero-knowledge proofs of correct decryption are known for lattice-based cryptography or other post-quantum encryption schemes. This state-of-affairs is unsatisfying, in particular because many applications that require zero-knowledge proofs of correct decryption should also be secure in the face of quantum computers which are becoming increasingly more powerful. For example, the electronic voting system Helios [Adi08] and the Estonian voting protocol [HW14] are using classical encryption schemes and decryption proofs with corresponding quantum threats to the long-term privacy of the voters.

On the contrary, there do exist efficient and straightforward passively secure lattice-based encryption schemes with distributed decryption. In such a scheme, the decryption key is shared among several players. Decryption is done in a distributed fashion by each player creating a decryption share, which can be individually verified, and a reconstruction algorithm can recover the message from the decryption shares. *Distributed decryption* allows more general methods to recover the message, such as general multi-party computation. There are many useful and efficient lattice-based threshold cryptosystems and distributed decryption schemes [BD10, BS13, DPSZ12, DOTT21, DHRW16, BKS19]. In particular, if the security requirements are relaxed, lattice-based distributed decryption can be very straight-forward.

Our main idea is to use MPC-in-the-head [IKOS07] in conjunction with a 2-party passively secure distributed decryption scheme to construct a very simple verifiable decryption scheme; however, we shall see that there are various technical challenges. To achieve the desired level of security, we run the 2-party decryption scheme on the ciphertexts many times locally, and then reveal a random subset of keys, one for each run, allowing others to verify that it was done correctly.

1.1 Contribution

Our main contribution is a transformation from a 2-party passively secure distributed decryption scheme to a 1-party verifiable decryption scheme. To achieve this, we use MPC-in-the-head with the 2-party decryption scheme. The idea is that the prover runs the 2-party decryption protocol many times and reveals the resulting decryption shares. The interactive verifier will then, for each run of the decryption scheme, ask to see one of the two decryption keys and any randomness involved in creating the corresponding decryption shares. With this information, it is straight-forward for the verifier to ensure that half of the decryption shares were generated honestly.

As usual, the idea is that if the prover cheats, the verifier will have probability (close to) 1/2 of detecting this in each round. If a cheating prover is consistently successful, we can use rewinding to extract both secret shares. Furthermore, if the 2-party decryption scheme is passively secure, revealing one share will not reveal anything about the secret key itself.

There are four remaining obstacles, two easy and two somewhat trickier. The first easy obstacle is that in a threshold public key encryption scheme or distributed decryption scheme, the decryption key shares are generated as part of key generation. We already have a decryption key, but we need to create many independent sharings of that key. For discrete logarithm-based schemes like ElGamal, this is usually trivial. For the schemes we consider, it is still not hard, but it follows that we do not have a fully general reduction from 2-party distributed decryption to (1-party) verifiable decryption. The second easy obstacle is that given both secret key shares we want to recover the secret key. We solve this by extending the notation of a distributed decryption function with a function which recovers the key from the shares. This is easy to satisfy in practice.

The third obstacle is that the verifier needs to make sure that the revealed key share is correct. For ordinary threshold decryption schemes, this can often be avoided, either because the dealer is trusted or replaced by some multi-party computation. Therefore, we need to use a non-generic solution here. For batched decryption, the main observation is that we only verify the key once for each run of the 2-party decryption scheme, not once per ciphertext in the batch. The number of runs essentially corresponds to the security parameter, which in many applications will be significantly smaller than the number of ciphertexts.

The final obstacle is related to our security proof. We need to simulate shares of the decryption key, any auxiliary information related to them, and decryption shares. Although similar techniques are common in the construction of threshold public key encryption scheme, the security definitions do not actually require their presence. Since we need them, our approach is again somewhat non-generic.

On the other hand, since we intend to verify correctness of decryption shares by revealing decryption key shares and any randomness involved, we can make do with a passively secure distributed decryption scheme, simplifying our work.

The result is a construction from a somewhat specialized 2-party distributed decryption scheme to a verifiable decryption scheme. Since the security requirements for the distributed decryption scheme are shifted compared to traditional threshold decryption schemes, this will allow us to use very simple threshold decryption. This means that it can be very efficient, both with respect to computational time and size of the decryption shares. Even though the decryption is run many times, the result will still be efficient compared to the alternatives.

Note that in an interactive setting, it may make sense to use a very small security parameter, making the protocol extremely cheap. For instance, in any system where detected cheating will have a significant penalty, rational actors will be deterred by even a small chance of detection. However, when the protocol is made non-interactive, this clearly does not work.

We prove in the interactive theorem prover Coq [BCHPM04] a simplified variant of our transform and an ElGamal toy example. Regrettably, we are unable to prove the full transform and the lattice example due to limitations in the interactive theorem prover. Indeed, to our knowledge, no interactive theorem prover exists which provides adequate support. Nevertheless, the proof of the simplified variant increases confidence in the result.

It is worth emphasizing that our protocol is very simple to implement (using Stern-based zero-knowledge proofs [KTX08, LNSW13] to ensure that key-shares are well-formed), lowering the bar for deploying our scheme in practice. We note that lattice-based zero-knowledge proofs in general can be very complicated, involving a combination of proofs of linear relations, proofs of shortness and range proofs, in addition to Gaussian sampling, rejection sampling and optimizations exploiting partially splitting rings and automorphisms [ALS20, LNS21]. Correctly and securely implementing voting systems using primitives based on discrete logarithms is hard [HLPT20], and lattice-based primitives makes it harder. In our protocol we only need to sample uniformly random or short elements in any ring of our choice, and use standard cut-and-choose techniques to open committed values, making it easy to use in practice. Concretely, this means that we are not vulnerable to side-channel attacks against Gaussian sampling [BHLY16] or rejection sampling [EFGT17].

Combined with the main contribution, this gives us a verifiable decryption scheme for a lattice-based public key encryption scheme that is very efficient when the number of ciphertexts is much larger than the security parameter. The protocol is fast and simple, and the proof size is small. We give concrete parameters and a proof of concept implementation of our protocol in Section 7.

1.2 Related Work

Verifiable decryption for ElGamal can be done by proving the equality of two discrete logarithms [CP92], and can be batched for significantly improved performance when decrypting many ciphertexts [Gor98, PBD07].

The "dual" Regev system [LPR13] can be used by making the randomness public. However, this is not zero-knowledge and opens for so-called "tagging-attacks" to de-anonymize users in privacy-preserving applications (e.g., e-voting).

Threshold encryption schemes [DF90] and distributed decryption schemes are now well-understood, and many constructions exist [BD10], in particular those related to SPDZ [DKL$^+$13, DPSZ12, KPR18]. When only passive security is required, these schemes can be quite efficient. Threshold decryption with active security implies verifiable decryption when the verification of decryption shares is a public operation. The problem is that it is often costly to provide a threshold decryption scheme with active security. Our approach gives away a decryption key share and randomness involved, and it is trivial to verify that the key share has been used correctly.

We compare more in detail with recently developed verifiable decryption protocols [BD10, BCOS20, LNS21, Sil22] in Section 8.

2 Passively Secure 2-party Decryption

A *distributed decryption scheme* enables a set of players to distribute the decryption of ciphertexts, in such a way that only authorized subsets of players can do the decryption. Usually, the decryption key shares are created once during key generation. As discussed in the introduction, we will generate independent decryption key sharings repeatedly, so we need to define the syntax of our variant of distributed decryption schemes precisely.

Consider a public key cryptosystem with key generation algorithm $\mathtt{KeyGen}$, encryption algorithm Enc and decryption algorithm Dec. We extend the notation with a predicate KeyM for key-matching which takes as input a public and secret key. We require for all matching public and secret keys pk, sk and all messages m, that $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$ (with overwhelming probability).

A *distributed decryption protocol* for this public key cryptosystem consists of four algorithms, a *dealer* algorithm, a *verify* algorithm, a *player* algorithm, and a *reconstruction* algorithm. We consider only two parties where both decrypt.

The dealer algorithm (Deal) takes as input a public key and corresponding secret key and outputs two *secret key shares* and some *auxiliary data* aux.
The verify algorithm (Verify) takes as input a public key, auxiliary data, an index and a secret key share and outputs *yes* (1) or *no* (0).
The player algorithm (Play) takes as input a secret key share and a ciphertext and outputs a *decryption share* ds.
The reconstruction algorithm (Rec) takes as input a ciphertext and two decryption shares and outputs either $\perp$ or a message.

Intuitively, the protocol is *correct* if Play and Rec collectively recover the encrypted message and verification accepts when the dealer is honest.

Definition 1 (Correctness). *A distributed decryption protocol is* correct *if for any key pair* $(\mathsf{pk}, \mathsf{sk})$ *s.t.* $\mathsf{KeyM}(\mathsf{pk}, \mathsf{sk}) = 1$, *all* $c = \mathsf{Enc}(\mathsf{pk}, m)$, *any* $(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{aux})$ *output by* $\mathsf{Deal}(\mathsf{pk}, \mathsf{sk})$, *then, for* $i = 0, 1$, $\mathsf{Verify}(\mathsf{pk}, \mathsf{aux}, i, \mathsf{sk}_i) = 1$, *and*

$$\Pr[\ m \leftarrow \mathsf{Dec}(\mathsf{sk}, c); \mathsf{Rec}(c, \mathsf{Play}(\mathsf{sk}_0, c), \mathsf{Play}(\mathsf{sk}_1, c)) = m\ ] \geq 1 - \mathsf{negl}.$$

For a distributed decryption protocol, we must trust the dealer for privacy, but not for integrity. The integrity property below says that if both secret shares given by the dealer are valid (according to the Verify algorithm), then the Play and Rec will collectively recover the encrypted message.

Definition 2 (Integrity). *A distributed decryption protocol has* integrity *if there exists an efficient algorithm (named* $\mathsf{FindKey}$ *which takes as input the public key, the two secret key shares and the auxiliary information, and returns a secret key) such that for all public keys pk, ciphertexts* $c = \mathsf{Enc}(\mathsf{pk}, m)$, *secret key shares* $(\mathsf{sk}_1, \mathsf{sk}_2)$, *and auxiliary data* aux *and* sk *output by* $\mathsf{FindKey}(\mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_1, \mathsf{aux})$ *satisfying* $\mathsf{Verify}(\mathsf{pk}, \mathsf{aux}, i, \mathsf{sk}_i) = 1$, *for* $i = 0, 1$, *we have that*

$$\Pr[\ \mathsf{KeyM}(\mathsf{pk}, \mathsf{sk}) \wedge \mathsf{Rec}(c, \mathsf{Play}(\mathsf{sk}_0, c), \mathsf{Play}(\mathsf{sk}_1, c)) = \mathsf{Dec}(\mathsf{sk}, c)\ ] \geq 1 - \mathsf{negl}.$$

For threshold cryptosystems and distributed decryption, security is typically defined through the usual security games for public key cryptosystem, allowing the adversary access to the decryption key shares through decryption share oracles. This security notion is not very convenient for us, so we shall instead rely on a variant of simulatability, namely we must be able to simulate both decryption key shares and decryption shares in a consistent fashion.

```
$Exp_{\mathcal{A}}^{ddp-sim-0}(\mathsf{pk}, \mathsf{sk})$
$(i, (c_0, ..., c_\tau), (m_0, ..., m_\tau)) \leftarrow \mathcal{A}(\mathsf{pk})$
$(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{aux}) \leftarrow \mathsf{Deal}(\mathsf{pk}, \mathsf{sk})$
$\forall j: \mathsf{ds}_j \leftarrow \mathsf{Play}(\mathsf{sk}_{1-i}, c_j)$
$b = \mathcal{A}(\mathsf{aux}, \mathsf{sk}_i, (\mathsf{ds}_0, ..., \mathsf{ds}_\tau))$
return $b$
```

```
$Exp_{\mathcal{A}}^{ddp-sim-1}(\mathsf{pk})$
$(i, (c_0, ..., c_\tau), (m_0, ..., m_\tau)) \leftarrow \mathcal{A}(\mathsf{pk})$
$(\mathsf{sk}_i, \mathsf{aux}) \leftarrow \mathsf{DealSim}(\mathsf{pk}, i)$
$\forall j: \mathsf{ds}_j \leftarrow \mathsf{PlaySim}(\mathsf{pk}, \mathsf{sk}_i, c_j, m_j)$
$b = \mathcal{A}(\mathsf{aux}, \mathsf{sk}_i, (\mathsf{ds}_0, ..., \mathsf{ds}_\tau))$
return $b$
```

Fig. 1. The passively secure experiment for distributed decryption protocols.

Definition 3 (Simulatability). *Consider a pair of algorithms* DealSim *and* PlaySim *and an adversary $\mathcal{A}$ playing the experiments from Figure 1, where $\mathcal{A}$ always outputs $\boldsymbol{c} = (c_0, ..., c_\tau), \boldsymbol{m} = (m_0, ..., m_\tau)$ such that $\{m_j = \mathsf{Dec}(\mathsf{sk}, c_j)\}_{j=1}^{\tau}$. The* simulatability advantage *of $\mathcal{A}$ is*

$$Adv^{ddp-sim}(\mathcal{A}, \mathsf{pk}, \mathsf{sk}) = \\ |\mathsf{Pr}[Exp_{\mathcal{A}}^{ddp-sim-0}(\mathsf{pk}, \mathsf{sk}) = 1] - \mathsf{Pr}[Exp_{\mathcal{A}}^{ddp-sim-1}(\mathsf{pk}) = 1]|,$$

where the probability is taken over the random tapes and $(\mathsf{pk}, \mathsf{sk})$ *output by* `KeyGen`*. We say that a distributed decryption protocol is* (t, ϵ)-simulatable *(or just* simulatable*) if no t-time algorithm $\mathcal{A}$ has advantage greater than ϵ.*

2.1 Toy Example: Distributed ElGamal

We briefly recall ElGamal encryption for a given cyclic group $\mathbb{G}$ of prime order p with generator g and give a toy decryption example.

Key generation (`KeyGen`) samples x from $\mathbb{Z}_p^*$ and return (g^x, x).

Encryption (Enc) takes as input a public key $\mathsf{pk} \in \mathbb{G}$ and message $\mathsf{m} \in \mathbb{G}$, samples r from $\mathbb{Z}_p^*$, and returns $(\mathsf{g}^r, \mathsf{pk}^r\mathsf{m})$.

Decryption (Dec) takes as input a secret key $x \in \mathbb{Z}_p^*$ and ciphertext $(\mathsf{c}_1, \mathsf{c}_2)$, and returns $\mathsf{c}_2/\mathsf{c}_1^x$.

Keymatch (KeyM) takes as input a public key $\mathsf{pk} \in \mathbb{G}$ and a secret key $x \in \mathbb{Z}_p^*$ and returns 1 if $\mathsf{g}^x = \mathsf{pk}$ and otherwise 0.

We will now give a distributed decryption protocol for ElGamal. This uses a $(2,2)$-secret sharing of the decryption key, and it works because of ElGamal's key-homomorphic property.

The dealer algorithm (Deal) takes as input a public key g^x and corresponding secret key x, samples x_0 from $\mathbb{Z}_p^*$, sets $x_1 = x - x_0$ and returns $(x_0, x_1, \mathsf{aux} = (\mathsf{g}^{x_0}, \mathsf{g}^{x_1}))$.

The verify algorithm (Verify) takes as input a public key pk, auxiliary data $\mathsf{aux} = (\mathsf{aux}_0, \mathsf{aux}_1)$, an index i and a secret key share x_i and outputs 1 iff $(\mathsf{g}^{x_i} = \mathsf{aux}_i) \wedge (\mathsf{pk} = \mathsf{aux}_0\mathsf{aux}_1)$.

The player algorithm (Play) takes as input a secret key share x_i and a ciphertext (c_1, c_2) and outputs *decryption share* $c_1^{x_i}$.

The reconstruction algorithm (Rec) takes as input a ciphertext $(\mathsf{c}_1, \mathsf{c}_2)$ and decryption shares (t_0, t_1) and outputs $\mathsf{c}_2/(t_0 t_1)$.

Correctness. Substituting ElGamal into the definition of correctness, for (g^x, x) and $(x_0, x_1, (g^{x_0}, g^{x_1})) \leftarrow \mathsf{Deal}(g^x, x)$, we get that the verify algorithm accepts both secret key shares and for any ciphertext $(g^r, g^{x^r} m)$, we get that

$$((g^x)^r m)/((g^r)^{x_0}(g^r)^{x_1}) = (g^r)^x m (g^r)^{-x_0-x_1} = m,$$

so correctness holds unconditionally.

Integrity. $\mathsf{FindKey}$ takes as input two key shares x_0, x_1 and outputs $x = x_0 + x_1$. Again, if the verify algorithm accepts both secret key shares, then we know that $g^x = g^{x_0} g^{x_1}$ and unconditional integrity follows from the computations above.

Privacy. Simulators $\mathsf{DealSim}$ and $\mathsf{PlaySim}$ work as follows:

– $\mathsf{DealSim}$ takes the public key pk and a bit i, samples x_i from $\mathbb{Z}_p^*$ and returns (wlog.) $(x_i, (\mathsf{g}^{x_i}, \mathsf{pk}/\mathsf{g}^{x_i}))$. It is clear that the auxiliary data and secret key from the simulator have the same distribution as the Deal.
– $\mathsf{PlaySim}$ takes as input public key pk, secret key x_i, ciphertext (c_1, c_2), and message m and returns a decryption share $c_2/(c_1^{x_i} m)$. Since m is the message encrypted in the ciphertext this is a perfect simulation if m is the correct decryption.

Note that these simulators are perfect due to ElGamal's elegant homomorphic structure, both with respect to keys and messages.

3 Verifiable Decryption from Distributed Decryption

We will now construct a (batch) zero-knowledge proof system of correct decryption from the distributed decryption protocol. The protocol is given in Figure 2. More precisely, our proof system is a sigma protocol with completeness, special soundness, and honest verifier-zero knowledge.

For any public key cryptosystem, a public key output by the key generation algorithm uniquely defines a decryption function that for all messages agrees with the decryption algorithm for any ciphertext output by the encryption algorithm, except those that lead to decryption failure.

Recall that for a batched verifiable decryption protocol the statement consists of a public key, a vector of ciphertexts and a vector of messages, where the ciphertexts have been output by the encryption algorithm. The statement is in the language if and only if the messages correspond to the decryption function applied to the ciphertexts. The secret key (witness) satisfies the relationship with the statement if it corresponds to the public key and the message vector is the decryption of the ciphertexts with the secret key.

The protocol works as follows: the prover creates λ sharings of the secret key by calling the Deal algorithm λ times. For each sharing and each ciphertext, the prover uses the Play algorithm to construct the decryption share. The prover sends the auxiliary information from Deal and all the shares to the verifier. Then, the verifier returns a challenge which is a binary vector of length λ. The prover finally reveals the corresponding parts of the shares as well as any randomness used in the Play algorithms with this key share. The prover checks that (1) all the revealed shares verify, (2) the decryption shares are consistent with the revealed key shares, and (3) the messages correspond to the decryption shares.

Completeness. Up to the possible negligible error introduced by decryption failures, completeness follows immediately by construction and the correctness of the underlying distributed decryption protocol.

Special Soundness. By rewinding, any cheating prover with a significant success probability can be used to create two accepting conversations $(w, \boldsymbol{\beta}, z)$ and $(w, \boldsymbol{\beta}', z')$, with $\boldsymbol{\beta} \neq \boldsymbol{\beta}'$. From this it follows that $\boldsymbol{\beta}[k] \neq \boldsymbol{\beta}'[k]$ for at least one k, and the verify algorithm has accepted both secret key shares and every decryption share in this round has been correctly created using the Play algorithm. Then, since the ciphertexts are encryptions of the first message vector, integrity implies that $\mathsf{FindKey}$ will recover a witness which matches the public key and for which the messages match the output of the decryption function.

Honest-Verifier Zero-Knowledge. Our simulator works as follows, given the statement $(\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau})$ and the challenge $\boldsymbol{\beta}$: First, for $i = 1, ..., \lambda$, we let $(\mathsf{aux}_i, \mathsf{sk}_{\boldsymbol{\beta}[i],i}) \leftarrow \mathsf{DealSim}(\mathsf{pk}, \boldsymbol{\beta}[i])$ and, for $j = 1, ..., \tau$, we let $\mathsf{ds}_{\boldsymbol{\beta}[i],j,i} \leftarrow \mathsf{PlaySim}(\mathsf{pk}, \mathsf{sk}_{\boldsymbol{\beta}[i],i}, c_i, m_i)$ and $\mathsf{ds}_{1-\boldsymbol{\beta}[i],j,i} \leftarrow \mathsf{Play}(\mathsf{pk}, \mathsf{sk}_{\boldsymbol{\beta}[i],i}, c_i)$. The proof transcripts is then $((\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau}), (\mathsf{aux}_i, \mathsf{ds}_{0,j,i}, \mathsf{ds}_{1,j,i}), \boldsymbol{\beta}, \mathsf{sk}_{\boldsymbol{\beta}[i],i})$. This is computationally indistinguishable from the honest transcripts if the distributed decryption protocol is simulatable.

4 Machine Checked Proofs

We adopt the definition of a sigma protocol from [HGT19] but do not require that the simulator always produces accepting transcripts when the statement is

```
$\Pi_{\text{ZKPCD}}$
Prover$((\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau}); (\mathsf{sk}))$          Verifier$(\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau})$
$k = 1, ..., \lambda:$
  $(\mathsf{sk}_{0,k}, \mathsf{sk}_{1,k}, \mathsf{aux}_k) \leftarrow \mathsf{Deal}(\mathsf{pk}, \mathsf{sk})$
  $i = 0, 1:$
    $j = 1, ..., \tau:$
      $\mathsf{ds}_{i,j,k} \leftarrow \mathsf{Play}(\mathsf{sk}_{i,k}, c_j; \rho_{i,k,j})$
$w \leftarrow (\{\mathsf{aux}_k, \{t_{i,j,k}\}\})$
                              $\xrightarrow{w}$
                                                  $\boldsymbol{\beta} \leftarrow_{\$} \{0,1\}^{\lambda}$
                              $\xleftarrow{\boldsymbol{\beta}}$
$z \leftarrow (\{\mathsf{sk}_{\boldsymbol{\beta}[k],k}\}_k, \{\rho_{\boldsymbol{\beta}[k],k,j}\}_{k,j})$
                              $\xrightarrow{z}$
                                                  $k = 1, ..., \lambda:$
                                                    $\mathsf{Verify}(\mathsf{pk}, \mathsf{aux}_k, \boldsymbol{\beta}[k], \mathsf{sk}_{\boldsymbol{\beta}[k],k}) \overset{?}{=} 1$
                                                    $j = 1, ..., \tau:$
                                                      $\mathsf{Play}(\mathsf{sk}_{\boldsymbol{\beta}[k],k}, c_j; \rho_{\boldsymbol{\beta}[k],k,j}) \overset{?}{=} \mathsf{ds}_{\boldsymbol{\beta}[k],j,k}$
                                                      $\mathsf{Rec}(c_j, \mathsf{ds}_{0,j,k}, \mathsf{ds}_{1,j,k}) \overset{?}{=} m_j$
```

Fig. 2. Proof of correct decryption. $\rho_{i,k,j}$ denotes the random tape used by the Play algorithm to create the ith share of the jth ciphertext in the kth run of the protocol.

not in the language. This does not affect on our intended use cases but prevents us from applying the standard transform to a disjunctive proof.

- We formally define an encryption scheme along the lines above in the paper but with perfect correctness. This can be found in the attached source in the Module Type EncryptionScheme.
- We formally define a distributed decryption schemes as a functor on encryptions schemes as above in the paper. However, we require perfect correctness, integrity and simulatability. (Module Type DistributedDecryption)
- We describe the transform for an arbitrary distributed decryption scheme and prove that the result is a sigma protocol for correct decryption. (Module ProofOfDecryption)
- We define the ElGamal cryptosystem and distributed decryption protocol and prove they satisfy the respective definitions. (ElGamal, DDElGamal).

The source code written in Coq is available online[†].

We are unable to do better because no interactive theorem provides good support for cryptographic arguments and support for lattice primitives. Nevertheless, our work is an important step in the direction of proving the full result if and when interactive theorem provers are ready.

5 Background: Lattice-Based Cryptography

5.1 The Cyclotomic Ring R_q

Let N be a power of 2 and q a prime such that $q \equiv 1 \mod 2N$. Then we define the rings $R = \mathbb{Z}[X]/\langle X^N + 1\rangle$ and $R_q = R/qR$, that is, R_q is the ring of polynomials modulo $X^N + 1$ with integer coefficients modulo q. This way, $X^N + 1$ splits completely into N irreducible factors modulo q, which allows for very efficient computation in R_q due to the number theoretic transform (NTT) [LN16].

We define the norms of elements

$$f(X) = \sum \alpha_i X^i \in R$$

to be the norms of the coefficient vector as a vector in $\mathbb{Z}^N$:

$$||f||_1 = \sum |\alpha_i|, \qquad ||f||_2 = \left(\sum \alpha_i^2\right)^{1/2}, \qquad ||f||_\infty = \max\{|\alpha_i|\}.$$

For an element $\bar{f} \in R_q$ we choose coefficients as the representatives in $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$, and then compute the norms as if $\bar{f}$ is an element in R. For vectors $\boldsymbol{a} = (a_1, \ldots, a_k) \in R^k$ we define the norms to be

$$\|\boldsymbol{a}\|_1 = \sum \|a_i\|_1, \qquad \|\boldsymbol{a}\|_2 = \left(\sum \|a_i\|_2^2\right)^{1/2}, \qquad \|\boldsymbol{a}\|_\infty = \max\{\|a_i\|_\infty\}.$$

5.2 Knapsack Problems

We first define the Search Knapsack problem in the ℓ_2 norm, also denoted as SKS^2. The SKS^2 problem is exactly the Ring-SIS problem in its Hermite Normal Form.

Definition 4. *The $\mathsf{SKS}^2_{N,q,\beta}$ problem is to find a short vector $\boldsymbol{x}$ of ℓ_2 norm less than or equal to β in R_q^2 satisfying $[\ a \quad 1\ ] \cdot \boldsymbol{x} = 0$ for a given uniformly random a in R_q. An algorithm $\mathcal{A}$ has advantage ϵ in solving the $\mathsf{SKS}^2_{N,q,\beta}$ problem if*

$$\Pr\left[\begin{array}{ll|c} [a & 1] \cdot \boldsymbol{x} = 0 & a \leftarrow\!\$\, R_q; \\ \wedge & \|x_i\|_2 \leq \beta & \boldsymbol{0} \neq \boldsymbol{x} \in R_q^2 \leftarrow \mathcal{A}(a) \end{array}\right] \geq \epsilon.$$

Additionally, we define the Decisional Knapsack problem in the ℓ_∞ norm denoted as DKS^∞. The DKS^∞ problem is equivalent to the Ring-LWE problem when the number of samples is limited.

[†] `www.dropbox.com/s/mn9gfmw3utkffyq`

Definition 5. *The* $\mathsf{DKS}^{\infty}_{N,q,\beta}$ *problem is to distinguish the distribution* $[\, a \quad 1 \,] \cdot \boldsymbol{x}$*, for a short* $\boldsymbol{x}$*, from the uniform distribution when given uniformly random* a *in* R_q*. An algorithm* $\mathcal{A}$ *has advantage* ϵ *in solving the* $\mathsf{DKS}^{\infty}_{N,q,\beta}$ *problem if*

$$\begin{aligned}&\big|\Pr[b = 1 \mid a \leftarrow\!\!\$\, R_q; \boldsymbol{x} \leftarrow\!\!\$\, R_q^2 \textit{ s.t. } \|\boldsymbol{x}\|_\infty \le \beta; b \leftarrow \mathcal{A}(a, [\, a \quad 1 \,] \cdot \boldsymbol{x})] \\ &- \Pr[b = 1 \mid a \leftarrow\!\!\$\, R_q; u \leftarrow\!\!\$\, R_q; b \leftarrow \mathcal{A}(a, u)]\big| \ge \epsilon.\end{aligned}$$

See [LM06, LPR10] for more details about knapsack problems.

5.3 BGV Encryption

We present a plain version of the BGV encryption scheme by Brakerski, Gentry and Vaikuntanathan [BGV12]. Let $p \ll q$ be primes, let R_q and R_p be polynomial rings modulo the primes q or p and $X^N + 1$ for a fixed N, let $B_\infty \in \mathbb{N}$ be a bound and let κ be the security parameter. The encryption scheme consists of three algorithms: key generation, encryption and decryption, where

- `KeyGen` samples an element $a \leftarrow\!\!\$\, R_q$ uniformly at random, samples short $s, e \leftarrow\!\!\$\, R_q$ such that $\max(\|s\|_\infty, \|e\|_\infty) \le B_\infty$. The algorithm outputs the public key $\mathsf{pk} = (a, b) = (a, as + pe)$ and the secret key $\mathsf{sk} = (s, e)$.
- `Enc`, on input the public key $\mathsf{pk} = (a, b)$ and an element m in R_p, samples short $r, e', e'' \leftarrow\!\!\$\, R_q$ such that the norm $\max(\|r\|_\infty, \|e'\|_\infty, \|e''\|_\infty) \le B_\infty$, and outputs the ciphertext $c = (u, v) = (ar + pe', br + pe'' + m)$ in R_q^2.
- `Dec`, on input the secret key $\mathsf{sk} = (s, e)$ and a ciphertext $c = (u, v)$, outputs the message $m = (v - su \mod q) \mod p$ in R_p.

The decryption algorithm is correct as long as the norm $\max\|v - su\|_\infty = B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$. It follows that the BGV encryption scheme is secure against chosen plaintext attacks if the $\mathsf{DKS}^{\infty}_{N,q,\beta}$ problem is hard for some $\beta = \beta(N, q, p, B_\infty)$.

Furthermore, we present the passively secure distributed decryption technique by Bendlin and Damgård [BD10] used in the MPC-protocols by Damgård *et al.* [DKL+13, DPSZ12]. When decrypting, we assume that each decryption server $\mathcal{D}_j$, for $1 \le j \le \xi$, has a uniformly random share $\mathsf{sk}_j = s_j$ of the secret key $\mathsf{sk} = (s, e)$ such that $s = s_1 + s_2 + ... + s_\xi$. Then they partially decrypt in the following way:

- `DistDec`, on input a secret key-share $\mathsf{sk}_j = s_j$ and a ciphertext $c = (u, v)$, computes $m_j = s_j u$, sample some large noise $E_j \leftarrow\!\!\$\, \mathbb{E} \subset R_q$ such that $\|E_j\|_\infty \le 2^{\mathtt{sec}}(B_{\mathtt{Dec}}/p\xi)$ for some statistical security parameter `sec` and upper error-bound $\max\|v - su\|_\infty \le B_{\mathtt{Dec}}$, then outputs $\mathsf{ds}_j = t_j = m_j + pE_j$.

We obtain the full decryption of the ciphertext (u, v) as $m \equiv (v - t \mod q) \mod p$, where $t = t_1 + t_2 + ... + t_\xi$. This will give the correct decryption as long as the noise $\max\|v - t\|_\infty \le (1 + 2^{\mathtt{sec}})B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$ (see [DKL+13, Appendix G]). Here, t will be indistinguishable from random except with probability $2^{-\mathtt{sec}}$.

5.4 Lattice-Based Commitments

We first define a commitment scheme and its security.

Definition 6 (Commitment Scheme). *A commitment scheme consists of three algorithms: key generation* (`KeyGen`)*, commitment* (`Com`) *and opening* (`Open`)*, where*

- `KeyGen`*, on input security parameter* 1^λ*, outputs public parameters* pp*,*
- `Com`*, on input message* m*, outputs commitment* c *and opening* r*,*
- `Open`*, on input* m*,* c *and* r*, outputs either* 0 *or* 1*,*

and the public parameters pp *are implicit inputs to* `Com` *and* `Open`*.*

Definition 7 (Completeness). *We say that the commitment scheme is* complete *if an honestly generated commitment is accepted by the opening algorithm. Hence, we want that*

$$\Pr\left[\mathtt{Open}(m,c,r)=1 \;:\; \begin{array}{c}\mathsf{pp}\leftarrow\mathtt{KeyGen}(1^\lambda)\\(c,r)\leftarrow\mathtt{Com}(m)\end{array}\right]=1,$$

where the probability is taken over the random coins of `KeyGen` *and* `Com`*.*

Definition 8 (Hiding). *We say that a commitment scheme is* hiding *if an adversary* $\mathcal{A}$*, after choosing two messages* m_0 *and* m_1 *and receiving a commitment* c *to either* m_0 *or* m_1 *(chosen at random), cannot distinguish which message* c *is a commitment to. Hence, we want that*

$$|\Pr\left[b=b' \;:\; \begin{array}{c}\mathsf{pp}\leftarrow\mathtt{KeyGen}(1^\lambda)\\(m_0,m_1,\mathsf{st})\leftarrow\mathtt{A}(\mathsf{pp})\\b\xleftarrow{\$}\{0,1\},c\leftarrow\mathtt{Com}(m_b)\\b'\leftarrow\mathtt{A}(c,\mathsf{st})\end{array}\right]-\frac{1}{2}|\leq\mathsf{negl},$$

where the probability is taken over the random coins of `KeyGen` *and* `Com`*.*

Definition 9 (Binding). *We say that a commitment scheme is* binding *if an adversary* $\mathcal{A}$*, after creating a commitment* c*, cannot find two valid openings to* c *for different messages* m *and* $\hat{m}$*. Hence, we want that*

$$\Pr\left[\begin{array}{c}m\neq\hat{m}\\\mathtt{Open}(m,c,r)=1\\\mathtt{Open}(\hat{m},c,\hat{r})=1\end{array} \;:\; \begin{array}{c}\mathsf{pp}\leftarrow\mathtt{KeyGen}(1^\lambda)\\(c,m,r,\hat{m},\hat{r})\leftarrow\mathtt{A}(\mathsf{pp})\end{array}\right]\leq\mathsf{negl},$$

where the probability is taken over the random coins of `KeyGen`*.*

We note that the public key in the BGV encryption scheme is essentially a commitment to the secret key. In general, ignoring the constant p, the value $b=as+e$ is a commitment to a short random secret s with randomness e if e is short and a is a uniformly random public element.

More formally, let q be a prime, let R_q be defined as above for a fixed N and let $B_\infty\in\mathbb{N}$ be a bound. These are the public parameters pp. The commitment scheme consists of three algorithms: key generation (`KeyGen`), commit (`Com`) and open (`Open`), where

- `KeyGen` samples an element $a' \leftarrow\$ R_q$ uniformly at random and outputs the public commitment key $\mathsf{pk}' = a'$.
- `Com`, on input the public key $\mathsf{pk}' = a'$ and a pseudo-random message m in R_q such that $\|m\|_\infty \leq B_\infty$, samples a short $r_m \leftarrow\$ R_q$ such that $\|r_m\|_\infty \leq B_\infty$, outputs commitment $c_m = a'm + r_m$ and opening $d_m = (m, r_m)$.
- `Open`, on input a commitment c_m and an opening $d_m = (m, r_m)$, checks if $\max(\|m\|_\infty, \|r_m\|_\infty) \leq B_\infty$ and $c_m = a'm + r_m$, and outputs 1 if both checks hold and otherwise 0.

It follows directly that the commitment scheme is (computationally) hiding if $\mathsf{DKS}^\infty_{N,q,B_\infty}$ is hard and (computationally) binding if $\mathsf{SKS}^2_{N,q,\sqrt{2N}\cdot B_\infty}$ is hard.

5.5 Zero-Knowledge Proof of Shortness

We present the Stern-based [Ste94] zero-knowledge proof of knowledge protocol by Kawachi *et al.* [KTX08] and Ling *et al.* [LNSW13]. We will later use this to prove that we know a valid opening d_m of a commitment c_m without leaking any information about the message nor the randomness. We denote the protocol by Π_{ZKPoS}.

Note that multiplication with a polynomial a' in R_q can be re-written as a matrix-vector product by a negacyclic $N \times N$-matrix $\boldsymbol{A}'$ over $\mathbb{Z}_q$. Define $\boldsymbol{A} = [\ \boldsymbol{A}' \quad \boldsymbol{I}_N\ ]$, and let $B_\infty = 1$ for simplicity (it can be generalized to any B_∞, see [LNSW13]). We give a zero-knowledge protocol for the following relation:

$$R_{\mathsf{DKS}^\infty_{N,q,1}} = \{((\boldsymbol{A}, \boldsymbol{y}); \boldsymbol{x})\colon \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \mod q \wedge \|\boldsymbol{x}\|_\infty = 1\}.$$

Furthermore, let B_m be the set of all vectors $\hat{\boldsymbol{x}}$ of length $3m$ with m coordinates of each element in $\{0, 1, -1\}$. It is then trivial to extend any witness $\boldsymbol{x}$ of the relation $R_{\mathsf{DKS}^\infty_{N,q,1}}$ to a vector $\hat{\boldsymbol{x}}$ in B_{2N} by appending values and extending the matrix $\boldsymbol{A}$ to the matrix $\hat{\boldsymbol{A}} = [\ \boldsymbol{A} \quad \boldsymbol{0}_{N\times 4N}\ ]$. It follows that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \mod q \wedge \|\boldsymbol{x}\|_\infty = 1$ if and only if $\hat{\boldsymbol{A}}\hat{\boldsymbol{x}} = \boldsymbol{y} \mod q \wedge \hat{\boldsymbol{x}} \in B_{2N}$. Let S_{6N} be the set of permutations on $6N$ symbols. The full protocol is given in Figure 3.

This protocol has soundness 2/3, and hence, must be repeated $\mu = \lambda \cdot \ln(2)/\ln(3/2)$ times to achieve soundness $2^{-\lambda}$. However, the protocol is very simple and lightweight in computation. We observe that the commitments c_1, c_2, c_3 are commitments to random values, and the commitments does not need any structure. Hence, we can compute the commitments as plain hashes of the committed values. Furthermore, we only need to sample a uniformly random permutation π from S_{6N} and a uniformly random vector $\boldsymbol{r}$ from $\mathbb{Z}_q^{6N}$. Compared to other lattice-based zero-knowledge protocols, there are no Gaussian sampling, no rejection sampling, no use of partially splitting rings or automorphisms.

We restrict permutations π to lie in the subgroup H generated by sign swaps and the transpositions $\{(i \quad i + 6N) \mid \text{for } i \text{ from 1 to } 6N\}$. This subgroup has 8^{6N} elements which can be represented by $18N$ bits. Each vector in $\mathbb{Z}_q^{6N}$ can be represented by $6N \log q$ bits, each vector in B_{2N} can be represented by $12N$ bits, and each commitment can be represented by 2κ bits. As β is uniformly

```
$\Pi_{\text{ZKPoS}}$

Prover$((\boldsymbol{A}, \boldsymbol{y}); \boldsymbol{x})$                          Verifier$(\boldsymbol{A}, \boldsymbol{y})$
expand $(\boldsymbol{A}, \boldsymbol{x})$ to $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{x}})$
$\pi \leftarrow\!\$\; S_{6N}$,    $\boldsymbol{r} \leftarrow\!\$\; \mathbb{Z}_q^{6N}$
$c_1 \leftarrow \texttt{Com}(\pi, \hat{\boldsymbol{A}}\boldsymbol{r})$
$c_2 \leftarrow \texttt{Com}(\pi(\boldsymbol{r}))$
$c_3 \leftarrow \texttt{Com}(\pi(\hat{\boldsymbol{x}} + \boldsymbol{r}))$
                          $\xrightarrow{w = (c_1, c_2, c_3)}$
                                                     $\beta \leftarrow\!\$\; \{1, 2, 3\}$
                          $\xleftarrow{\beta}$
if $\beta = 1$:
    $z = (\boldsymbol{s} = \pi(\hat{\boldsymbol{x}}), \boldsymbol{t} = \pi(\boldsymbol{r}))$
if $\beta = 2$:
    $z = (\phi = \pi, \boldsymbol{u} = \hat{\boldsymbol{x}} + \boldsymbol{r})$
if $\beta = 3$:
    $z = (\psi = \pi, \boldsymbol{v} = \boldsymbol{r})$
                          $\xrightarrow{z}$          $\Pi_{\text{ZKPoSV}}$:
                                                     if $\beta = 1$:
                                                         parse $z$ as $\boldsymbol{s}$ and $\boldsymbol{t}$
                                                         if $c_2 \overset{?}{=} \texttt{Com}(\boldsymbol{t})$ and $c_3 \overset{?}{=} \texttt{Com}(\boldsymbol{s} + \boldsymbol{t})$ and $\boldsymbol{s} \in B_{2N}$
                                                             return 1
                                                     if $\beta = 2$:
                                                         parse $z$ as $\phi$ and $\boldsymbol{u}$
                                                         if $c_1 \overset{?}{=} \texttt{Com}(\phi, \hat{\boldsymbol{A}}\boldsymbol{u} - \boldsymbol{y})$ and $c_3 \overset{?}{=} \texttt{Com}(\phi(\boldsymbol{u}))$
                                                             return 1
                                                     if $\beta = 3$:
                                                         parse $z$ as $\psi$ and $\boldsymbol{v}$
                                                         if $c_1 \overset{?}{=} \texttt{Com}(\psi, \hat{\boldsymbol{A}}\boldsymbol{v})$ and $c_2 \overset{?}{=} \texttt{Com}(\psi(\boldsymbol{v}))$
                                                             return 1
                                                     return 0
```

Fig. 3. Zero-knowledge proof of shortness.

distributed we estimate the proof size assuming that each response will appear a third of the times each. The total proof, denoted π_S, is of size

$$|\pi_S| = (6\kappa + 16N + 6N \log q)\mu \text{ bits.} \tag{1}$$

We finally note that the protocol can be improved using the combinatorial extensions by Beullens [Beu20], reducing the size of the proof by a factor 10 without much extra computational work nor increased complexity in the implementation.

6 Zero-Knowledge Protocol of Correct Decryption

6.1 Lattice-Based Distributed Decryption

Setup. We will be working over the ring $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$ together with a modulus $p \ll q$, both prime. These are the public parameters of the protocol, together with security parameter κ, soundness parameter λ, bound B_∞ and maximal ciphertext error-bound B_{Dec}. We define commitments, their security and give a concrete instantiation based on lattices in the full version of this paper. The commitments are both computationally hiding and computationally binding, in addition to being linearly homomorphic. Finally, let $(\Pi_{\text{ZKPoS}}, \Pi_{\text{ZKPoSV}})$ be a non-interactive zero-knowledge protocol for the following relation:

$$R_{\mathsf{DKS}^{\infty}_{N,q,1}} = \{((\boldsymbol{A}, \boldsymbol{y}); \boldsymbol{x})\colon \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \mod q \wedge \|\boldsymbol{x}\|_\infty = 1\}.$$

Scheme. We present a distributed decryption version of the BGV encryption scheme [BGV12], where KeyGen, Enc and Dec are defined in Section 5.3.

The dealer algorithm (Deal) takes as input a public key $\mathsf{pk} = (a, b)$ and corresponding secret key $\mathsf{sk} = (s, e)$, samples uniform s_0 and e_0 from R_q, and computes $s_1 = s - s_0$ and $e_1 = e - e_0$. Then it commits to the values as $c_{s_i} = \mathtt{Com}(s_i), c_{e_i} = \mathtt{Com}(e_i)$, and computes $b_i = as_i + pe_i$ so that $b = b_0 + b_1$. Finally, it computes non-interactive zero-knowledge proofs π_{S_i} proving that the sums $s_0 + s_1$ and $e_0 + e_1$ are short (see details in Section 7). It outputs key shares $sk_0 = (s_0, e_0), sk_1 = (s_1, e_1)$ and $\mathsf{aux} = (b_0, b_1, c_{s_0}, c_{s_1}, c_{e_0}, c_{e_1}, \pi_{S_0}, \pi_{S_1})$.

The verify algorithm (Verify) takes as input a public key $\mathsf{pk} = (a, b)$, an index i, a secret key share $\mathsf{sk}_i = (s_i, e_i)$, openings d_{s_i} and d_{e_i}, and aux. It outputs 1 if and only if $(b_i \overset{?}{=} as_i + pe_i) \wedge (b \overset{?}{=} b_0 + b_1) \wedge \mathtt{Open}(c_{s_i}, d_{s_i}) \wedge \mathtt{Open}(c_{e_i}, d_{e_i}) \wedge (\Pi_{\text{ZKPoSV}}(\mathsf{sk}_i, \mathsf{aux}, \pi_{S_i}))$, and 0 otherwise.

The player algorithm (Play) takes as input a key share $\mathsf{sk}_i = (s_i, e_i)$, a ciphertext $c = (u, v)$, samples bounded E_i and outputs $\mathsf{ds}_i = t_i = s_i u + pE_i$.

The reconstruction algorithm (Rec) takes as input a ciphertext $c = (u, v)$, decryption shares (t_0, t_1), and outputs $m = (v - t_0 - t_1 \mod q) \mod p$.

6.2 Security

Theorem 1 (Correctness). *The distributed decryption scheme in 6.1 is correct with respect to Definition 1 when* $\max\|v - t\|_\infty \leq (1 + 2^{\mathsf{sec}})B_{\mathsf{Dec}} < \lfloor q/2 \rfloor$.

Theorem 2 (Integrity). *Suppose the protocol Π_{ZKPoS} is (computationally) sound and that* `Com` *is (computationally) binding. Let $\mathcal{A}_0$ be an adversary against integrity of the distributed decryption scheme with advantage ϵ_0, and let λ be the number of rounds in the protocol. Then there exists adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ against soundness of Π_{ZKPoS} and binding of* `Com`*, respectively, with advantages ϵ_1 and ϵ_2, such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2 + 2^{-\lambda}$. The runtime of $\mathcal{A}_1$ and $\mathcal{A}_2$ are essentially the same as the runtime of $\mathcal{A}_0$.*

Proof. We sketch the argument. There are essentially three possible ways to attack the integrity of the protocol: an attacker that knows the secret decryption key but correctly guess the challenge in each round is able to decrypt to arbitrary messages, and otherwise, if the attacker does not know the secret key, needs to break the underlying schemes. The guessing attack has success probability $2^{-\lambda}$.

For Verify to accept for both $i = 0$ and $i = 1$, we need that $b = b_0 + b_1$, $b_0 = as_0 + pe_0$, $b_1 = as_1 + pe_1$ and that the zero-knowledge proof of shortness π_S of the sums $s_0 + s_1$ and $e_0 + e_1$ are accepted. If either of the key shares are incorrect then Verify accept with probability 0, and if the key shares are correct, then Rec outputs m except with negligible probability. An attacker can choose s_0, s_1, e_0 and e_1 such that all equations are correct, but the sums are not short. The soundness of Verify then reduces to the soundness of the zero-knowledge protocol, and an attacker $\mathcal{A}_0$ against this part of the protocol with advantage ϵ_0 can be turned into an attacker $\mathcal{A}_1$ against Π_{ZKPoS} with the same advantage.

The last option is for the attacker to produce commitments to a true but unrelated statement with respect to the secret key used in the encryption scheme. This allows the attacker to produce a valid proof of shortness without cheating, but for an unrelated key. However, Verify only accepts if both the opening of the commitments are correct and the zero-knowledge proof of shortness verifies. Hence, and attacker $\mathcal{A}_0$ that is able to produce valid openings and proofs with advantage ϵ_0 can be turned into an attacker $\mathcal{A}_2$ against `Com` with the same advantage by rewinding the prover for the zero-knowledge proof of knowledge of short openings and then extract two different but valid openings to the commitment. □

Theorem 3 (Privacy). *Suppose the protocol Π_{ZKPoS} is (statistically) honest-verifier zero-knowledge, that* `Com` *is (computationally) hiding and that* Enc *is (computationally) CPA secure. Then there exists a simulator for the verifiable decryption protocol such that for any distinguisher $\mathcal{A}_0$ for this simulator with advantage ϵ_0 there exists an adversary $\mathcal{A}_2$ against hiding for the commitment scheme with advantage ϵ_2, an adversary $\mathcal{A}_3$ against CPA security for the encryption scheme with advantage ϵ_3, and a distinguisher $\mathcal{A}_1$ for the simulator of Π_{ZKPoS} with advantage ϵ_1, such that $\epsilon_0 \leq \epsilon_1 + \epsilon_2 + \epsilon_3$. The runtime of $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are essentially the same as the runtime of $\mathcal{A}_0$.*

Proof. Let $\texttt{Sim}_{\text{Short}}$ be a simulator for Π_{ZKPoS}. We present a simulator $\mathsf{DealSim}$ for the Deal-algorithm and a simulator $\mathsf{PlaySim}$ for the Play-algorithm in Figure 4.

```
$\mathsf{DealSim}(\mathsf{pk} = (a, b), i)$
$i = 0, 1\colon s_i^* \leftarrow\!\!\$\ R_q, \quad e_i^* \leftarrow\!\!\$\ R_q$
$b_i^* = as_i^* + pe_i^*, \quad b_{1-i}^* = b - b_i^*$
$c_{s_i}^* \leftarrow \texttt{Com}(s_i^*), c_{s_{1-i}}^* \leftarrow \texttt{Com}(s_{1-i})$
$c_{e_i}^* \leftarrow \texttt{Com}(e_i^*), c_{e_{1-i}}^* \leftarrow \texttt{Com}(s_{1-i})$
$\pi_S^* \leftarrow \texttt{Sim}_{\text{Short}}(c_{s_i}^*, c_{s_{1-i}}^*, c_{e_i}^*, c_{e_{1-i}}^*)$
$\mathsf{aux}^* \leftarrow (b_0^*, b_1^*, c_{s_0}^*, c_{s_1}^*, c_{e_0}^*, c_{e_1}^*, \pi_S^*)$
$\textbf{return } (\mathsf{sk}_i^* = (s_i^*, e_i^*), \mathsf{aux}^*)$
```

```
$\mathsf{PlaySim}(\mathsf{sk}_{1-i} = (s_{1-i}, e_{1-i}), c = (u, v), i, m)$
$E_{1-i} \leftarrow\!\!\$\ \mathbb{E}$
$t_{1-i} = s_{1-i}u + pE_{1-i}$
$t_i^* = v - m - t_{1-i} \mod p$
$\textbf{return } (\mathsf{ds}_i^* = t_i^*)$
```

Fig. 4. Simulators $\mathsf{DealSim}$ and $\mathsf{PlaySim}$.

$\mathsf{DealSim}$: We create the simulator in three steps. We first replace π_S by the simulated proof π_S^* produced by $\texttt{Sim}_{\text{Short}}$. An attacker $\mathcal{A}_0$ with advantage ϵ_0 against this change can be turned into an attacker $\mathcal{A}_1$ against the simulator $\texttt{Sim}_{\text{Short}}$ of protocol Π_{ZKPoS} with the same advantage.

Next, we replace the key shares by uniformly random key-shares s_i^* and e_i^* that give correctness, that is, the public key-shares b_0^* and b_1^* sum to b, but s_0^* and s_1^* does not need to sum to a short key s^* and e_0^* and e_1^* does not need to sum to short noise e^*. This ensures that Verify outputs 1. An attacker $\mathcal{A}_0$ with advantage ϵ_0 against this change can then be turned into an attacker $\mathcal{A}_3$ against CPA security of the encryption scheme with the same advantage.

Finally, we replace the commitments to unopened values by commitments to random values. This way, none of the values in the protocol any longer depends on the secret key in the protocol, and b_i^* are simulated perfectly. An attacker $\mathcal{A}_0$ with advantage ϵ_0 against this change can then be turned into an attacker $\mathcal{A}_2$ against hiding of the commitment scheme with the same advantage.

$\mathsf{PlaySim}$: we start by sampling bounded E_{1-i} from $\mathbb{E}$ and computing $t_{1-i} = s_{1-i}u + pE_{1-i}$. Then we find t_i such that $(v - t_0 - t_1 \mod q) \mod p = m$. This ensures that Rec outputs the message m when reconstructing the shares. Here, the values are sampled according to the exact same distribution as in the real protocol, and the statistical distance is negligible in the security parameter κ.

□

6.3 Zero-Knowledge Proof of Verifiable Decryption

We present the different phases of our sigma protocol for proving correct decryption. The protocol is given in Figure 5. The security of the construction follows directly from the results in Section 3 in combination with Theorem 1, 2 and 3.

Setup. We are given a honestly generated public key $\mathsf{pk} = (a, b = as + pe)$, where $\max(\|s\|_\infty, \|e\|_\infty) \leq B_\infty$. The secret key $\mathsf{sk} = (s, e)$ is given to the prover. We are given a set of honestly generated ciphertexts $\{(u_j, v_j) = (ar_j + pe'_j, br_j + pe''_j + m_j)\}_{j=1}^{\tau}$, where $\max(\|r\|_\infty, \|e'\|_\infty, \|e''\|_\infty) \leq B_\infty$, and set of messages $\{m_j\}_{j=1}^{\tau}$.

Commit phase. For soundness parameter λ, the prover does the following for $k = 1, ..., \lambda$. First, it runs the Deal algorithm on sk and pk to produce $\mathsf{sk}_{0,k}, \mathsf{sk}_{1,k}$ and aux_k. It uses Π_{ZKPoS} to prove that the shares are correctly computed. Then, for $i = 0, 1$ and each $j = 1, ..., \tau$, it runs the Play algorithm on each key-share $\mathsf{sk}_{i,k}$ and ciphertext c_j to produce $t_{0,j,k}$ and $t_{1,j,k}$. Finally, it sends $w \leftarrow (\{\mathsf{aux}_k, \{t_{i,j,k}\}_{i=0,j=1}^{1,\tau}\}_{k=1}^{\lambda})$ to end the commitment phase.

Challenge phase. The verifier independently samples a random binary challenge vector $\boldsymbol{\beta}$ of length λ. It sends $\boldsymbol{\beta}$ to the prover.

Respond phase. The prover sends openings $z = (\{d_{s_{\boldsymbol{\beta}[k],k}}, d_{e_{\boldsymbol{\beta}[k],k}}\})$, for each of the commitments to each index k of $\boldsymbol{\beta}$, to the verifier.

Verification phase. For each $k = 1, ..., \lambda$, the verifier runs the Verify algorithm to make sure that the openings of $s_{\boldsymbol{\beta}[k],k}$ and $e_{\boldsymbol{\beta}[k],k}$ are valid, check that all shares of the public key are computed correctly as $b_{\boldsymbol{\beta}[k],k} = as_{\boldsymbol{\beta}[k],k} + pe_{\boldsymbol{\beta}[k],k}$, verify the public key $b = b_{0,k} + b_{1,k}$ and ensure that each $\pi_{S_{i,k}}$ is valid. Further, for each $j = 1, ..., \tau$, the verifier runs the Rec algorithm to make sure that all decryption shares are correct and that all messages are decrypted correctly. It outputs 1 if all checks hold, and 0 otherwise.

Fiat-Shamir. To make the scheme non-interactive we can use the Fiat-Shamir transform [FS87] by hashing the output of the commit phase and use the hash as challenge, before outputting the response. We note that this can be done similarly to the optimizations described for estimating the size in the next section. We also note that the soundness parameter λ initially can be very small in the interactive case, while it should be (approximately) as large at the security parameter κ in the non-interactive setting, increasing the size of the proof of decryption.

Hybrid proof. We note that the interaction in the protocol opens for a hybrid proof: if we wish for a quick result to get confidence in the decrypted ciphertexts but at the same time can wait longer to be completely certain, we can ask for two proofs. First, we ask the prover for a proof where $\lambda_I = 10$ or $\lambda_I = 20$, and sample a random challenge ourselves. If we accept the proof, we ask the prover to compute a non-interactive proof for the same statement but with $\lambda_N = 100$. This proof can be received, stored and verified later, knowing already that the messages most likely are correctly decrypted. The interactive proof also allows the verifier to arbitrarily increase λ_I by sending more challenges on the fly, where we tell the prover when we are done, and he creates the proofs of shortness in the end. This is particularly useful in real-world applications, e.g., e-voting.

```
$\Pi_{\text{ZKPCD}}$
Prover$(((a,b),\{(u_j,v_j)\}_{j=1}^{\tau},\{m_j\}_{j=1}^{\tau});(s,e))$                Verifier$((a,b),\{(u_j,v_j)\}_{j=1}^{\tau},\{m_j\}_{j=1}^{\tau})$
$k = 1,\ldots,\lambda:$
  Deal:
  $(s_{0,k}, s_{1,k}) \leftarrow\$ \langle s\rangle$
  $(e_{0,k}, e_{1,k}) \leftarrow\$ \langle e\rangle$
  $i = 0,1:$
    $(c_{s_{i,k}}, d_{s_{i,k}}) \leftarrow \texttt{Com}(s_{i,k})$
    $(c_{e_{i,k}}, d_{e_{i,k}}) \leftarrow \texttt{Com}(e_{i,k})$
    $b_{i,k} \leftarrow as_{i,k} + pe_{i,k}$
    $j = 1,\ldots,\tau:$
    Play:
      $E_{i,j,k} \leftarrow\$ \mathbb{E}$
      $t_{i,j,k} = s_i u_j + pE_{i,j,k}$
    $\pi_{S_{0,k}} \leftarrow \Pi_{\text{ZKPoS}}(c_{s_{0,k}}, c_{s_{1,k}}); (d_{s_{0,k}}, d_{s_{1,k}}))$
    $\pi_{S_{1,k}} \leftarrow \Pi_{\text{ZKPoS}}(c_{e_{0,k}}, c_{e_{1,k}}); (d_{e_{0,k}}, d_{e_{1,k}}))$
$w \leftarrow (\{b_{i,k}, c_{s_{i,k}}, c_{e_{i,k}}, \pi_{S_{i,k}}, \{t_{i,j,k}\}_j\}_{i,k})$
                                   $\xrightarrow{w}$
                                                        $\boldsymbol{\beta} \leftarrow\$ \{0,1\}^{\lambda}$
                                   $\xleftarrow{\boldsymbol{\beta}}$
$z \leftarrow (\{d_{s_{\boldsymbol{\beta}[k],k}}, d_{e_{\boldsymbol{\beta}[k],k}}\}_k)$
                                   $\xrightarrow{z}$
                                                        $k = 1,\ldots,\lambda:$
                                                          Verify:
                                                          $\texttt{Open}(c_{s_{\boldsymbol{\beta}[k],k}}, d_{s_{\boldsymbol{\beta}[k],k}}) \overset{?}{=} 1$
                                                          $\texttt{Open}(c_{e_{\boldsymbol{\beta}[k],k}}, d_{e_{\boldsymbol{\beta}[k],k}}) \overset{?}{=} 1$
                                                          $1 \overset{?}{\leftarrow} \Pi_{\text{ZKPoSV}}(c_{s_{0,k}}, c_{s_{1,k}}, \pi_{S_{0,k}})$
                                                          $1 \overset{?}{\leftarrow} \Pi_{\text{ZKPoSV}}(c_{e_{0,k}}, c_{e_{1,k}}, \pi_{S_{1,k}})$
                                                          $b_{\boldsymbol{\beta}[k],k} \overset{?}{=} as_{\boldsymbol{\beta}[k],k} + pe_{\boldsymbol{\beta}[k],k}$
                                                          $b \overset{?}{=} b_{0,k} + b_{1,k}$
                                                          $j = 1,\ldots,\tau:$
                                                          Rec:
                                                            $pE_{\boldsymbol{\beta}[k],j,k} = t_{\boldsymbol{\beta}[k],j,k} - u_j s_{\boldsymbol{\beta}[k],k}$
                                                            $\|pE_{\boldsymbol{\beta}[k],j,k}\|_\infty \overset{?}{\leq} 2^{\mathsf{sec}-1} B_{\mathsf{Dec}}$
                                                            $v_j - t_{0,j,k} - t_{1,j,k} \overset{?}{\equiv}_p m_j$
```

Fig. 5. Zero-knowledge proof of correct decryption.

7 Performance

In this section, we shall carefully analyze the performance of our decryption proof. Along the way, we make several easy optimizations with respect to the protocol in Figure 5. In particular, we use a commitment in the first message, and then send only the values that the verifier cannot recompute himself in the second message. Finally, we compute the zero-knowledge proofs of shortness in the response phase instead of the commit phase, reducing the number of proofs by a factor of two in each round of the protocol.

7.1 Proof Size

Each element in R_q is of size $N \log q$ bits, which might be large, and each element in R_p is of size $N \log p$ bits, which will be small. Short elements bounded by B_∞ is of size $N \log B_\infty$ bits. We let $\mathtt{H}$ be a collision resistant hash-function with output of length 2κ. Note that the soundness parameter λ may be chosen independently of, and in particular smaller than, the security parameter κ.

Commit phase. To reduce the number of ring elements being sent, we commit to the output of the commit phase using a hash-function, and send the hash instead. More concretely, we let $w = \mathtt{H}(\{b_{0,k}, b_{1,k}, c_{s_{0,k}}, c_{s_{1,k}}, c_{e_{0,k}}, c_{e_{1,k}}, \{t_{i,j,k}\}_{i=0,j=1}^{1,\tau}\}_{k=1}^{\lambda})$.

Challenge phase. The verifier sends the vector $\boldsymbol{\beta}$ consisting of λ independently sampled bits to the prover.

Respond phase. Note that we do not need to send the partial decryptions $t_{\boldsymbol{\beta}[k],j,k}$, because they can be computed uniquely from u_j, $s_{\boldsymbol{\beta}[k],k}$ and $E_{\boldsymbol{\beta}[k],j,k}$, and we can let a uniform binary seed $\rho_{\boldsymbol{\beta}[k],k}$ of length 2κ bits can be used to deterministically generate the randomness used in each round. Next, we also note that $b_{\boldsymbol{\beta}[k],k}$ can be computed directly from $s_{\boldsymbol{\beta}[k],k}$ and $e_{\boldsymbol{\beta}[k],k}$, and $b_{1-\boldsymbol{\beta}[k],k}$ from b and $b_{\boldsymbol{\beta}[k],k}$.

It follows that, for each $k = 1, ..., \lambda$, the prover sends $s_{\boldsymbol{\beta}[k],k}$ and $e_{\boldsymbol{\beta}[k],k}$, commitments $c_{s_{1-\boldsymbol{\beta}[k],k}}$ and $c_{e_{1-\boldsymbol{\beta}[k],k}}$ together with the openings $d_{s_{\boldsymbol{\beta}[k],k}}$ and $d_{e_{\boldsymbol{\beta}[k],k}}$, and the partial decryptions $\{t_{1-\boldsymbol{\beta}[k],j,k}\}_{j=1}^{\tau}$. Since the commitments to the sharings of s and e are used in the zero-knowledge proof of shortness, these commitment is computed using lattice-based commitments. We observe that $c_{s_k} = c_{s_{1-\boldsymbol{\beta}[k],k}} + \mathtt{Com}(s_{\boldsymbol{\beta}[k],k})$ and $c_{e_k} = c_{e_{1-\boldsymbol{\beta}[k],k}} + \mathtt{Com}(e_{\boldsymbol{\beta}[k],k})$, with randomness zero, are commitments to $s_{\boldsymbol{\beta}[k],k} + s_{1-\boldsymbol{\beta}[k],k}$ and $e_{\boldsymbol{\beta}[k],k} + e_{1-\boldsymbol{\beta}[k],k}$, which are short. Then we use the zero-knowledge proof of shortness to prove that we know openings of c_{s_k} and c_{e_k} to get π_{S_0} and π_{S_1}. Denote all proofs of shortness by π_S.

Total communication. The total proof size sent by the prover is

$$2\kappa + \lambda N(4 \log q + 2\kappa + 2 \log B_\infty) + \lambda \tau N \log q + |\pi_S| \text{ bits.}$$

Zero-knowledge proof of shortness. There are many options for π_S, proving knowledge of valid openings of the commitments c_{s_k} and c_{e_k}. We can use the Fiat-Shamir with aborts framework [Lyu09, Lyu12], but this would give us a large soundness slack, that is, we prove knowledge of a vector that might be much larger than what we started with. This would increase the parameters to be used in the overall protocol. Other alternatives are the exact proofs using MPC-in-the-head techniques by Baum and Nof [BN20] or the range proofs by Attema *et al.* [ALS20]. However, we note that even though these are efficient, both protocols are very complex and are complicated to implement correctly for use in the real world. Another approach is to use generic proof systems such as Ligero [AHIV17] or Aurora [BCR$^+$19], adding more complexity to the overall protocol. We can also use the amortized proof by Bootle *et al.* [BBC$^+$18] to prove that all λ executions are done correctly at the same time. This is the most efficient proof system for these relations today.

However, assuming that the soundness parameter λ is much smaller than the number of ciphertexts τ, the size of the proofs of shortness does not matter much. To keep the protocol as simple as possible, to make it easier to implement the protocol and avoid bugs in practice, we choose to use the Stern-based proofs by Kawachi *et al.* [KTX08] and Ling *et al.* [LNSW13] in our implementation and estimates.

Concrete parameters. For a concrete instantiation, we use the example parameters in Table 1, estimated to $\kappa = 128$ bits of long-term security using the LWE-estimator [APS15] with the *BKZ.qsieve* cost-model. Inserting these parameters into the proof of shortness, then each proof $\pi_{S_{i,k}}$ is of size $\approx 87\mu$ KB. This makes $|\pi_S| \approx 175\mu\lambda$ KB. Furthermore, using the improvements by Beullens [Beu20] we can shrink the proofs down to $18\mu\lambda$ KB. If we replace π_S with the amortized proof by Bootle *et al.* [BBC$^+$18] we get a proof of total size 520 KB*. However, if the number of ciphertexts τ is very large, we can ignore all other terms and get a proof of correct decryption π_D of size $\approx 14\lambda\tau$ KB. See Table 1 for details. The ciphertext modulus q is chosen to be large enough to ensure correct decryption.

7.2 Implementation

We wrote a proof of concept implementation of our scheme in C++ using the NTL-library [Sho21]. The implementation was benchmarked on an Intel Core i5 running at 2.3 GHz with 16 GB RAM. We ran the protocol with $\lambda = 40, \tau = 1000, \mu = 68$. The timings are given in Table 1. The implementation is very simple, and consists of a total of 400 lines of code. Our source code is available online **. We note that our implementation does not use the number theoretic transform for fast multiplication of elements in the ring to reduce complexity. A rough comparison to NFLlib [ABG$^+$16], where they show clear improvements

* Setting $m = 2048, \log q = 55, r = 90, b = 3, \tau = 50, k = 2398, l = 5000$ and $h = 100$ for soundness 2^{-45} and run the protocol twice, see [BBC$^+$18, Section 4.1] for details.

** github.com/tjesi/verifiable-decryption-in-the-head.

Parameter	Explanation	Constraints	Value
N	Dimension	Power of two	2048
q	Ciphertext modulus	$B_{\texttt{Dec}} \ll q \equiv 1 \mod 2N$	$\approx 2^{55}$
p	Plaintext modulus		2
κ	Security parameter	Long-term privacy	128
$\texttt{sec}$	Statistical security		40
λ	Soundness parameter		$10, ..., 128$
μ	Repetitions of Π_{ZKPoS}	$\mu \geq \lambda \cdot \ln(2)/\ln(3/2)$	$17, ..., 218$
B_∞	Bounds on secrets		1
$B_{\texttt{Dec}}$	Decryption bound	$\|v - su\|_\infty \leq B_{\texttt{Dec}}$	$\approx 2^{13}$

Size of π_D	Timings for π_D	Size of π_S	Timings for π_S
$14\lambda\tau$ KB	$4\lambda\tau$ ms	$175\lambda\mu$ KB	$30\lambda\mu$ ms

Table 1. Notation, explanation, constraints and concrete parameters for the protocol. We also provide size and timings for decryption proof π_D and proofs of shortness π_S.

compared to NTL, indicates that an optimized implementation should provide a speedup by at least an order of magnitude.

8 Comparison

8.1 Comparison to DistDec (TCC'10)

We sketch an extension of the passively secure distributed decryption protocol Π_{DistDec} given by Bendlin and Damgård [BD10], which is used in SPDZ [DKL⁺13, DPSZ12]. The main difference compared to our protocol is that this protocol requires zero-knowledge proofs to ensure correct computation at each step of the protocol to achieve active security instead of repeating the decryption procedure several times. The protocol works roughly as following:

1. Each party $\mathcal{D}_i$ samples uniform $E_{i,j}$ such that $\|E_{i,j}\|_\infty \leq 2^{40} B_{\texttt{Dec}}/\xi p$ (for 40 bits statistical security) and computes the partial decryptions $t_{i,j} = s_i u_j + pE_{i,j}$ for each ciphertext $c_j = (u_j, v_j)$.
2. Each party $\mathcal{D}_i$ publish a zero-knowledge proof $\pi_{L_{i,j}}$ of the linear relation for $t_{i,j}$, using the lattice-based commitments together with their zero-knowledge proof of linear relations by Baum *et al.* [BDL⁺18].
3. Each party $\mathcal{D}_i$ use the amortized proof by Baum *et al.* [BBC⁺18] for size N to prove that each $E_{i,j}$ is bounded by $2^{\texttt{sec}} B_{\texttt{Dec}}/\xi p$, for commitments $\boldsymbol{c}_{E_{i,j}}$.
4. The verifier checks the relations $(v_j - t_{0,j} - t_{1,j} \mod q) \equiv m_j \mod p$ and that all the zero-knowledge proofs are valid.

Elements t_j and commitments $\boldsymbol{c}_{E_{i,j}}$ are $N \log q$ and $2N \log q$ bits, respectively. Each proof of linearity $\pi_{L_{i,j}}$ is $6N \log(6\bar{\sigma})$ bits. The amortized proof is $540 \log(6\hat{\sigma})$

bits. The total size, for each $\mathcal{D}_i$, is

$$(3N \log q + 6N \log(6\bar{\sigma}) + 540 \log(6\hat{\sigma}))\tau \text{ bits.}$$

Then one party can split the key into $\xi = 2$ shares, run Π_{DistDec} on each key-share locally, and return the outputs from both $\mathcal{D}_1$ and $\mathcal{D}_2$ together with an additional proof that the key-splitting was correct. We based the estimate on the parameters from Table 1, with $\bar{\sigma} \approx 2^{16}$ and $\hat{\sigma} \approx 2^{66}$ (see e.g. Aranha *et al.* [ABGS22] for details about proofs and sizes). However, the amortized proof is not exact, which means that we must increase q to $q \approx 2^{78}$ to ensure correct decryption. For security $\kappa = 128$ we also need to increase N to $N = 4096$. The proof is then of size $\approx 363\tau$ KB. We conclude that Π_{ZKPCD} is of equal size as Π_{DistDec} for $\lambda = 26$ and otherwise larger.

We do not have access to timings for this protocol. However, as the modulus is much larger, the dimension is twice the size, the zero-knowledge proofs include Gaussian sampling and rounds of aborts, we expect the protocol to be much slower than ours despite the large number of repetitions in our construction.

8.2 Comparison to Boschini *et al.* (PQ Crypto'20)

Boschini *et al.* [BCOS20] presents a zero-knowledge protocol for Ring-SIS and Ring-LWE. Their protocol can be used to prove knowledge of secrets or plaintexts, or prove correct decryption given a message and a BGV ciphertext. Concrete estimates for the latter are not given in the paper, but the number of constraints is higher for decryption than for the former. For a slightly smaller choice of parameters, a single proof of plaintext knowledge is of size 87 KB and takes roughly 3 minutes to compute. We conclude that the proof system by Boschini *et al.* will provide decryption proofs of equal size as protocol when $\lambda = 6$ and smaller otherwise. The time it takes to produce such a proof are several orders of magnitude slower than ours, making the system impossible to use in practice even for moderate sized sets of ciphertexts.

8.3 Comparison to Lyubashevsky *et al.* (PKC'21)

A recent publication by Lyubashevsky, Nguyen and Seiler [LNS21] gives a verifiable decryption protocol for the Kyber encapsulation scheme [SAB^{+}20]. Here, the encryption is over a rank 2 module over a ring of dimension $N = 256$ and modulus $q = 3329$ with secret and noise values bounded by $B_\infty = 2$. The proof of correct decryption of binary messages of dimension 256 is of size 43.6 KB, which of equal size as in our protocol for $\lambda = 3$. We note that the message space is smaller than in our protocol, mostly because we are forced to choose larger parameters to ensure correct decryption, and hence, we can not provide a proof of verifiable decryption for Kyber in particular. They do not provide timings, but we notice that the proof system use Gaussian sampling, rejection sampling, partially splitting rings and automorphisms – making the protocol very difficult to implement correctly and securely in practice.

8.4 Comparison to Silde (VOTING'22)

Silde [Sil22] presents a direct verifiable decryption of BGV ciphertexts. The parameters are similar to our scheme, and the proof is of 43.6 KB per ciphertext. This the same as in our scheme for $\lambda = 3$, ignoring the setup cost, while smaller for larger λ. The timing of the decryption protocol is 76 ms per ciphertext, which is equal to our timings for $\lambda = 19$ and otherwise up to 7 times faster for $\lambda = 128$.

9 Conclusion and Future Work

9.1 Summary and Conclusion

We have defined a passively secure distributed decryption protocol, and show how this can be used to construct an interactive zero-knowledge protocol for correct decryption. This is the first both efficient and simple single-party verifiable decryption protocol for lattice-based cryptography when instantiated with the BGV encryption scheme.

The size and efficiency of the protocol is a small factor times $\lambda\tau$, for arbitrary soundness parameter λ and number of ciphertexts τ. The long-term privacy parameter of the protocol κ can be set independently of, and in particular larger than, λ. This allows an interactive instantiation of the protocol to be very efficient, both in size and computation. For $\kappa = 128$ we estimate the decryption proof to be of size $\approx 14\lambda\tau$ KB and the proof/verification time to be only 4λ ms per ciphertext, when τ is much larger than λ.

Altogether, our new lattice-based proof of decryption provides a unique combination of efficiency and simplicity that make our proof system an interesting candidate for real-world applications.

9.2 Future Improvements and Extensions

Remove the ZK-proofs of shortness. The Deal-algorithm outputs a zero-knowledge proof proving that the sum of the shares of the secret key and noise used to compute the public key are short. This is to ensure the correctness and security of the encryption scheme. However, ElGamal does not require such a proof, and it might be infeasible to find key-shares that are correct, but not short, that decrypts consistently for all BGV-ciphertexts. We would need to conduct a more careful analysis to ensure that our construction is secure also without the zero-knowledge proofs.

Instantiations based on other primitives. A natural future step is to apply our transformation to other encryption schemes, also with other underlying hardness assumptions. As an example, a threshold scheme has was recently constructed based on isogenies [DM20].

Thanks

We thank Carsten Baum and the anonymous reviewers for helpful comments.

References

ABG$^+$16. Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. NFLlib: NTT-based fast lattice library. In Kazue Sako, editor, *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 341–356. Springer, Heidelberg, February / March 2016.

ABGS22. Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. Cryptology ePrint Archive, Report 2022/422, 2022. `https://ia.cr/2022/422`.

Adi08. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008: 17th USENIX Security Symposium*, pages 335–348. USENIX Association, July / August 2008.

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 2087–2104. ACM Press, October / November 2017.

ALS20. Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler. Practical product proofs for lattice commitments. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 470–499. Springer, Heidelberg, August 2020.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

BBC$^+$18. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 669–699. Springer, Heidelberg, August 2018.

BCHPM04. Yves Bertot, Pierre Castéran, Gérard Huet, and Christine Paulin-Mohring. *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions.* Texts in theoretical computer science. Springer, 2004.

BCOS20. Cecilia Boschini, Jan Camenisch, Max Ovsiankin, and Nicholas Spooner. Efficient post-quantum SNARKs for RSIS and RLWE and their applications to privacy. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 247–267. Springer, Heidelberg, 2020.

BCR$^+$19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, Heidelberg, May 2019.

BD10. Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 201–218. Springer, Heidelberg, February 2010.

BDL$^+$18. Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 368–385. Springer, Heidelberg, September 2018.

Beu20. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 183–211. Springer, Heidelberg, May 2020.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325. Association for Computing Machinery, January 2012.

BHLY16. Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 323–345. Springer, Heidelberg, August 2016.

BKS19. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 3–33. Springer, Heidelberg, May 2019.

BN20. Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 495–526. Springer, Heidelberg, May 2020.

BS13. Slim Bettaieb and Julien Schrek. Improved lattice-based threshold ring signature scheme. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 34–51. Springer, Heidelberg, June 2013.

CP92. David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.

DF90. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, Heidelberg, August 1990.

DHRW16. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016,* *Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122. Springer, Heidelberg, August 2016.

DKL$^+$13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, September 2013.

DM20. Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 187–212. Springer, Heidelberg, May 2020.

DOTT21. Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 99–130. Springer, Heidelberg, May 2021.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, Heidelberg, August 2012.

EFGT17. Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1857–1874. ACM Press, October / November 2017.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, Heidelberg, August 1987.

Gor98. Daniel M. Gordon. A Survey of Fast Exponentiation Methods. *J. Algorithms*, 27(1):129–146, 1998.

HGT19. Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified verifiers for verifying elections. In *CCS*, pages 685–702. ACM, 2019.

HLPT20. Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy*, pages 644–660. IEEE Computer Society Press, May 2020.

HM20. Thomas Haines and Johannes Müller. SoK: Techniques for verifiable mix nets. In Limin Jia and Ralf Küsters, editors, *CSF 2020: IEEE 33st Computer Security Foundations Symposium*, pages 49–64. IEEE Computer Society Press, 2020.

HW14. Sven Heiberg and Jan Willemson. Verifiable internet voting in Estonia. In *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014*, 2014.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 21–30. ACM Press, June 2007.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189. Springer, Heidelberg, April / May 2018.

KTX08. Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 372–389. Springer, Heidelberg, December 2008.

LM06. Vadim Lyubashevsky and Daniele Micciancio. Generalized compact Knapsacks are collision resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006: 33rd International Colloquium on Automata, Languages and Programming, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 144–155. Springer, Heidelberg, July 2006.

LN16. Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16: 15th International Conference on Cryptology and Network Security*, volume 10052 of *Lecture Notes in Computer Science*, pages 124–139. Springer, Heidelberg, November 2016.

LNS21. Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 215–241. Springer, Heidelberg, May 2021.

LNSW13. San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 107–124. Springer, Heidelberg, February / March 2013.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, Heidelberg, May / June 2010.

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 35–54. Springer, Heidelberg, May 2013.

LW18. Fucai Luo and Kunpeng Wang. Verifiable decryption for fully homomorphic encryption. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *ISC 2018: 21st International Conference on Information Security*, volume 11060 of *Lecture Notes in Computer Science*, pages 347–365. Springer, Heidelberg, September 2018.

Lyu09. Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616. Springer, Heidelberg, December 2009.

Lyu12. Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755. Springer, Heidelberg, April 2012.

PBD07. Kun Peng, Colin Boyd, and Ed Dawson. Batch zero-knowledge proof and verification and its applications. *ACM Trans. Inf. Syst. Secur.*, 10(2):6, 2007.

SAB$^+$20. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

Sho21. Victor Shoup. Ntl: A library for doing number theory, 2021. `https://libntl.org/index.html`.

Sil22. Tjerand Silde. Verifiable Decryption for BGV. Workshop on Advances in Secure Electronic Voting, 2022. `https://ia.cr/2021/1693`.

SSA$^+$18. Fatemeh Shirazi, Milivoj Simeonovski, Muhammad Rizwan Asghar, Michael Backes, and Claudia Diaz. A survey on routing in anonymous communication protocols. *ACM Comput. Surv.*, 51(3), June 2018.

Ste94. Jacques Stern. A new identification scheme based on syndrome decoding. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 13–21. Springer, Heidelberg, August 1994.

Paper V

Short Paper: Verifiable Decryption for BGV

Tjerand Silde

Accepted at the Workshop on Advances in Secure Electronic Voting, VOTING 2022. The paper is available at: eprint.iacr.org/2021/1693.pdf

Short Paper: Verifiable Decryption for BGV

Tjerand Silde

Department of Mathematical Sciences
Norwegian University of Science and Technology
tjerand.silde@ntnu.no

Abstract. In this work we present a direct construction for verifiable decryption for the BGV encryption scheme by combining existing zero-knowledge proofs for linear relations and bounded values. This is one of the first constructions of verifiable decryption protocols for lattice-based cryptography, and we give a protocol that is simpler and at least as efficient as the state of the art when amortizing over many ciphertexts. To prove its practicality we provide concrete parameters, resulting in proof size of less than 44τ KB for τ ciphertexts with message space 2048 bits. Furthermore, we provide an open source implementation showing that the amortized cost of the verifiable decryption protocol is only 76 ms per message when batching over $\tau = 2048$ ciphertexts.

Keywords: lattice cryptography · verifiable decryption · zero-knowledge

1 Introduction

Many privacy preserving applications require one to prove that a ciphertext is correctly decrypted without revealing the secret key. This is called *verifiable decryption*, formalized by Camenisch and Shoup [CS03]. Example use-cases are electronic voting [Adi08], mixing networks [HM20], DC-networks [CWF13] and fully homomorphic encryption [LW18]. These applications usually require decrypting a large number of ciphertexts.

Unfortunately, the above systems are either not secure against quantum computers or very inefficient. Recent works in lattice-based cryptography are leading towards voting protocols achieving security even against quantum adversaries, see, e.g., the shuffles by Aranha *et al.* [ABG$^+$21], Costa *et al.* [CMM19] and Farzaliyev *et al.* [FWK21]. However, few constructions provides verifiable decryption for lattice-based encryption schemes.

1.1 Contribution

We present a new and efficient verifiable decryption protocol for batches of ciphertext using the lattice-based encryption scheme by Brakerski, Gentry and Vaikuntanathan [BGV12]. The protocol is direct; the decryption procedure consists of computing a linear equation involving the ciphertext and the key, and then the message is extracted by rounding the result modulo the plaintext moduli. This procedure gives the correct result if the noise-level in the ciphertext is bounded. We use lattice-based commitments to commit to the secret key, and then we prove two relations in zero-knowledge: 1) we prove that the linear equation holds with respect to a fresh commitment to the ciphertext-noise, and 2) prove that the noise is bounded. Together, this leads to an efficient verifiable decryption protocol. We give concrete parameters and estimate the size in Sec. 4.1 and give timings from our proof-of-concept implementation in Sec. 4.2.

1.2 Related Work

We compare to the works on verifiable decryption for lattices by Lyubashevsky *et al.* [LNS21], Gjøsteen *et al.* [GHM+22] and Boschini *et al.* [BCOS20] in Sec 4.3.

2 Lattice-Based Cryptography

Let N be a power of 2 and $q = 1 \mod 2N$ a prime. We define the ring $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$. For $f \in R_q$ we choose coefficients as the representatives in $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$, and compute inner products $\langle\cdot,\cdot\rangle$ and norms as vectors over $\mathbb{Z}$:

$$\|f\|_1 = \sum |\alpha_i|, \qquad \|f\|_2 = \left(\sum \alpha_i^2\right)^{1/2}, \qquad \|f\|_\infty = \max\{|\alpha_i|\}.$$

We furthermore define the sets $S_{\beta_\infty} = \{x \in R_q \mid \|x\|_\infty \leq \beta_\infty\}$ as well as

$$\mathcal{C} = \{c \in R_q \mid \|c\|_\infty = 1, \|c\|_1 = \nu\} \text{ and } \bar{\mathcal{C}} = \{c - c' \mid c \neq c' \in \mathcal{C}\}.$$

2.1 Rejection Sampling

We want to output vectors $\boldsymbol{z} = \boldsymbol{y} + \boldsymbol{v}$ such that $\boldsymbol{z}$ is independent of $\boldsymbol{v}$, and hence, $\boldsymbol{v}$ is masked by the vector $\boldsymbol{y}$. If $\boldsymbol{y}$ is sampled according to a Gaussian distribution $\mathcal{N}_\sigma^k$ with standard deviation σ, then we want $\boldsymbol{z}$ to be from the same distribution. $1/M$ is the success probability for rejection sampling, and M is computed as

$$\max \frac{\mathcal{N}_\sigma^k(\boldsymbol{z})}{\mathcal{N}_{\boldsymbol{v},\sigma}^k(\boldsymbol{z})} = \exp\left[\frac{-2\langle \boldsymbol{z}, \boldsymbol{v}\rangle + \|\boldsymbol{v}\|_2^2}{2\sigma^2}\right] \leq \exp\left[\frac{24\sigma\|\boldsymbol{v}\|_2 + \|\boldsymbol{v}\|_2^2}{2\sigma^2}\right] = M,$$

so that $|\langle \boldsymbol{z}, \boldsymbol{v}\rangle| < 12\sigma\|\boldsymbol{v}\|_2$ with probability at least $1 - 2^{-100}$. Hence, for $\sigma = 11\|\boldsymbol{v}\|_2$, we get $M \approx 3$. This is the standard way to choose parameters. If the procedure is only done once for the vector $\boldsymbol{v}$, we can decrease the parameters, to the cost of leaking only one bit of information about $\boldsymbol{v}$ from the given $\boldsymbol{z}$.

Lyubashevsky *et al.* [LNS21] suggest to require that $\langle \boldsymbol{z}, \boldsymbol{v}\rangle \geq 0$. Then we can set $M = \exp(\|v\|_2/2\sigma^2)$. For $\sigma = 0.675\|\boldsymbol{v}\|_2$, we get $M \approx 3$, with the effect of rejecting about half of the vectors up front. See [LNS21, Figure 2] for details.

2.2 Hardness Assumptions

We first define the Search Knapsack problem in the ℓ_2 norm, also denoted as SKS^2. The SKS^2 problem is the Ring-SIS problem in its Hermite Normal Form.

Definition 1. *The $\mathsf{SKS}^2_{N,q,\beta}$ problem is to find a short vector $\boldsymbol{x}$ of ℓ_2 norm less than or equal to β in R_q^2 satisfying $[\, a \quad 1\,] \cdot \boldsymbol{x} = 0$ for a given uniformly random a in R_q. An algorithm $\mathcal{A}$ has advantage ϵ in solving the $\mathsf{SKS}^2_{N,q,\beta}$ problem if*

$$\Pr\left[\begin{array}{l} [a \quad 1] \cdot \boldsymbol{x} = 0 \\ \wedge \quad \|x_i\|_2 \leq \beta \end{array} \;\middle|\; \begin{array}{c} a \leftarrow\!\$\, R_q; \\ \boldsymbol{0} \neq \boldsymbol{x} \in R_q^2 \leftarrow \mathcal{A}(a) \end{array}\right] \geq \epsilon.$$

We also define the Decisional Knapsack problem (DKS^∞) in the ℓ_∞ norm. DKS^∞ is equivalent to the Ring-LWE problem when the number of samples is limited.

Definition 2. *The $\mathsf{DKS}^\infty_{N,q,\beta_\infty}$ problem is to distinguish the distribution $[\, a \quad 1\,] \cdot \boldsymbol{x}$, for a short $\boldsymbol{x}$, from the uniform distribution when given uniformly random a in R_q. An algorithm $\mathcal{A}$ has advantage ϵ in solving the $\mathsf{DKS}^\infty_{N,q,\beta_\infty}$ problem if*

$$\begin{aligned} &|\Pr[b = 1 \mid a \leftarrow\!\$\, R_q; \boldsymbol{x} \leftarrow\!\$\, S_{\beta_\infty}; b \leftarrow \mathcal{A}(a, [\, a \quad 1\,] \cdot \boldsymbol{x})] \\ &- \Pr[b = 1 \mid a \leftarrow\!\$\, R_q; u \leftarrow\!\$\, R_q; b \leftarrow \mathcal{A}(a, u)]| \geq \epsilon. \end{aligned}$$

See [LM06, LPR10] for more details about knapsack problems over rings.

2.3 BGV Encryption

Let $p \ll q$ be primes, let R_q and R_p be defined as above for a fixed N, let $\mathtt{D}$ be a bounded distribution over R_q, let $\beta_\infty \in \mathbb{N}$ be a bound and let λ be the security parameter. The (plain) BGV encryption scheme [BGV12] consists of three algorithms: key generation (KGen), encryption (Enc) and decryption (Dec), where

- KGen samples $a \leftarrow\!\$\, R_q$ uniformly at random, samples a short $s \leftarrow\!\$\, S_{\beta_\infty}$ and samples noise $e \leftarrow \mathtt{D}$. It outputs keys $\mathsf{pk} = (a, b) = (a, as + pe)$ and $\mathsf{sk} = s$.
- Enc, on input pk and a message m in R_p, samples a short $r \leftarrow\!\$\, S_{\beta_\infty}$, samples noise $e', e'' \leftarrow \mathtt{D}$, and outputs ciphertext $c = (u, v) = (ar + pe', br + pe'' + m)$.
- Dec, on input $\mathsf{sk} = s$ and $c = (u, v)$, outputs $m = (v - su \mod q) \mod p$.

The decryption is correct if $\max\|v - su\|_\infty = B_{\mathtt{Dec}} < \lfloor q/2 \rfloor$. The encryption scheme is CPA-secure if the $\mathsf{DKS}^\infty_{N,q,\beta}$ problem is hard for some $\beta = \beta(N, q, p, \beta_\infty)$.

2.4 Lattice-Based Commitments

Let $\mathcal{N}_{\sigma_\mathrm{C}}$ be a Gaussian distribution over R_q with standard deviation σ_C. The commitment scheme by Baum *et al.* [BDL+18] consists of three algorithms: key generation (KGen), committing (Com) and opening (Open), where

- KGen outputs a public key $\texttt{pk}$ to commit to messages in R_q. We define

$$\begin{aligned} \boldsymbol{A}_1 &= \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{A}'_1 \end{bmatrix} && \text{where } \boldsymbol{A}'_1 \leftarrow\!\!\$\; R_q^{n \times (k-n)} \\ \boldsymbol{a}_2 &= \begin{bmatrix} \boldsymbol{0}^n & 1 & \boldsymbol{a}'_2 \end{bmatrix} && \text{where } \boldsymbol{a}'_2 \leftarrow\!\!\$\; R_q^{(k-n-1)}, \end{aligned}$$

  for height $n+1$ and width k and let $\texttt{pk}$ be $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 \\ \boldsymbol{a}_2 \end{bmatrix}$.
- Com commits to messages $m \in R_q$ by sampling $\boldsymbol{r}_m \leftarrow\!\!\$\; S_{\beta_\infty}$, and computes

$$\texttt{Com}_{\texttt{pk}}(m; \boldsymbol{r}_m) = \boldsymbol{A} \cdot \boldsymbol{r}_m + \begin{bmatrix} \boldsymbol{0} \\ m \end{bmatrix} = \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} = [\![m]\!].$$

  Com outputs commitment $[\![m]\!]$ and opening $\boldsymbol{d} = (m, \boldsymbol{r}_m, 1)$.
- Open verifies whether $(m, \boldsymbol{r}_m, f)$, with $f \in \bar{\mathcal{C}}$, is a valid opening of $[\![m]\!]$ with respect to $\texttt{pk}$ by checking that $\|\boldsymbol{r}_m[i]\|_2 \leq 4\sigma_{\mathrm{C}}\sqrt{N}$, for $i \in [k]$, and if

$$f \cdot \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} \stackrel{?}{=} \boldsymbol{A} \cdot \boldsymbol{r}_m + f \cdot \begin{bmatrix} \boldsymbol{0} \\ m \end{bmatrix}.$$

  Open outputs 1 if all these conditions holds, and 0 otherwise.

The commitment scheme is hiding if the $\mathsf{DKS}^{\infty}_{N,q,\beta_\infty}$ problem is hard and it is binding if the $\mathsf{SKS}^{2}_{N,q,16\sigma_{\mathrm{C}}\sqrt{\nu N}}$ problem is hard, see [BDL+18, Section 4].

2.5 Zero-Knowledge Proof of Linear Relations

Let $[\![y]\!], [\![y']\!]$ be commitments such that $y' = \alpha y + \beta$ for some public values $\alpha, \beta \in R_q$. The protocol Π_{LIN} in [ABG+21, Figure 1] is a zero-knowledge proof of knowledge, with ℓ_2 bound $B_{\mathrm{C}} = 2\sigma_{\mathrm{C}}\sqrt{N}$ on the responses $\boldsymbol{z}_i$, for the relation:

$$\mathcal{R}_{\mathrm{Lin}} = \left\{ (x, w) \;\middle|\; \begin{array}{l} x = (\alpha, \beta, [\![y]\!], [\![y']\!]), w = (y, \boldsymbol{r}_y, \boldsymbol{r}_{y'}, f, f') : \\ \texttt{Open}([\![y]\!], y, \boldsymbol{r}_y, f) = \texttt{Open}([\![y']\!], \alpha \cdot y + \beta, \boldsymbol{r}_{y'}, f') = 1 \end{array} \right\}$$

When applying the Fiat-Shamir transform [FS87], we let the challenge $c \in \mathcal{C}$ be the output of a hash function applied to the full transcript. Then, we get the proof $\pi_L = (c, \boldsymbol{z}_1, \boldsymbol{z}_2)$, where each $\boldsymbol{z}_i$ is of size $kN \log_2(6\sigma_{\mathrm{C}})$ bits. We can compress each $\boldsymbol{z}_i$ to get a proof of total size $2(k-n)N \log_2(6\sigma_{\mathrm{C}})$ bits by checking an approximate equality instead [ABG+21, Section 3.2]. We denote by

$$\begin{aligned} \pi_L &\leftarrow \Pi_{\mathrm{LIN}}((y, \boldsymbol{r}_y, \boldsymbol{r}_{y'}, f_y, f_{y'}); (\alpha, \beta, [\![y]\!], [\![y']\!])), \text{ and} \\ 0 \vee 1 &\leftarrow \Pi_{\mathrm{LINV}}((\alpha, \beta, [\![y]\!], [\![y']\!]); \pi_L), \end{aligned}$$

the run of the proof and verification protocols, respectively, where the verification protocol Π_{LinV} performs the checks as in the last step in [ABG+21, Figure 1] and also verifies that c was computed correctly with respect to the transcript. Π_{Lin} is a sound proof of knowledge in the ROM if the $\mathsf{SKS}^{2}_{N,q,2B_{\mathrm{C}}}$ problem is hard.

2.6 Amortized Zero-Knowledge Proof of Bounded Openings

Let $\boldsymbol{A}$ be a publicly known $r \times v$-matrix over R_q, let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_\tau$ be bounded elements in R_q^v and let $\boldsymbol{A}\boldsymbol{s}_i = \boldsymbol{t}_i$ for $i \in [\tau]$. Letting $\boldsymbol{S}$ be the matrix whose columns are $\boldsymbol{s}_i$ and $\boldsymbol{T}$ be the equivalent matrix for $\boldsymbol{t}_i$, Baum *et al.* [BBC+18] give a efficient amortized zero-knowledge proof of knowledge for the relation:

$$\mathcal{R}_{\mathrm{A}} = \left\{ (x, w) \;\middle|\; \begin{array}{c} x = (\boldsymbol{A}, \boldsymbol{T}), w = \boldsymbol{S}: \\ \forall i \in [\tau]: \; \boldsymbol{t}_i = \boldsymbol{A}\boldsymbol{s}_i \wedge ||\boldsymbol{s}_i||_2 \leq 2 \cdot B_{\mathrm{A}} \end{array} \right\}$$

The protocol Π_{A} is depicted in [BBC+18, Figure 1]. We use a challenge matrix $\boldsymbol{C}$ with entries sampled from $\mathcal{C}_{\mathrm{A}} = \{0, 1\}$. For security parameter λ, we define the number of parallel protocol instances to be $\hat{n} = \lambda + 2$. Denote by

$$\pi_{\mathrm{A}} \leftarrow \Pi_{\mathrm{A}}(\boldsymbol{S}; (\boldsymbol{A}, \boldsymbol{T})), \text{ and } 0 \vee 1 \leftarrow \Pi_{\mathrm{AV}}((\boldsymbol{A}, \boldsymbol{T}); \pi_{\mathrm{A}}),$$

the run of the proof and verification protocols, respectively, where the Π_{A}-protocol, using Fiat-Shamir, produces a proof of the form $\pi_{\mathrm{A}} = (\boldsymbol{C}, \boldsymbol{Z})$, where $\boldsymbol{C}$ is the output of a hash-function applied to the full transcript, and the Π_{AV}-protocol consists of the two checks in the last step in [BBC+18, Figure 1]. The verification bound on each column of $\boldsymbol{Z}$ is $B_{\mathrm{A}} = \sqrt{2N}\sigma_{\mathrm{A}}$. Note that σ_{A}, and also B_{A}, depends on the norm of $\boldsymbol{S}$ (see rejection sampling in Section 2.1). Hence, the bound we can prove depends on the number of equations in the statement. Π_{A} is a sound proof of knowledge in the ROM if $\mathsf{SKS}^2_{N,q,2B_{\mathrm{A}}}$ is hard.

3 The Verifiable Decryption Protocol

The protocol is direct. The prover starts by decrypting the ciphertext (u, v) to obtain the underlying plaintext m as $m = (v - us \mod q) \mod p$. Then, he commits to the noise d $(= er + e'' - se')$ in the ciphertexts as $[\![d]\!]$. Finally, he proves two statements in zero-knowledge: 1) the linear relation $p[\![d]\!] = v - m - u[\![s]\!]$ holds modulo q with respect to the noise and a public commitment to the secret key, and 2) the value committed to in $[\![d]\!]$ is shorter than some bound $B < q/2p$.

More precisely, we present a proof protocol for the following relation:

$$R_{\mathrm{DEC}} = \left\{ (x, w) \;\middle|\; \begin{array}{l} x = ((a, b), [\![s]\!], (u_1, v_1), \ldots, (u_\tau, v_\tau), m_1, \ldots, m_\tau), \\ w = (s, \boldsymbol{r}_s, f_s) \text{ such that } \mathtt{Open}([\![s]\!]; s, \boldsymbol{r}_s, f_s) = 1 \\ \wedge\ \forall i \in [\tau]: \; pd_i = v_i - m_i - u_i s \;\wedge\; \|d_i\|_\infty < q/2p. \end{array} \right\}$$

Here, we assume that either a trusted dealer generated the public key and secret key together with a commitment to the secret key, or that the prover already has proved in zero-knowledge that the public key is well formed and that the secret key is committed to in $[\![s]\!]$, using any exact proof from the literature.

The verifiable decryption protocol Π_{DEC}, for prover $\mathcal{P}$, goes as following:

1. $\mathcal{P}$ takes as input a set of ciphertexts $(u_1, v_1), \ldots, (u_\tau, v_\tau)$ and $([\![s]\!], s, \boldsymbol{r}_s, f_s)$.
2. $\mathcal{P}$ runs $\mathtt{Dec}$ on input s and (u_i, v_i) for all $i \in [\tau]$ to obtain messages $m_1, \ldots, m_\tau$.

3. $\mathcal{P}$ extracts noise d_i by computing $d_i = (v_i - m_i - u_i s)/p \mod q$ for all $i \in [\tau]$.
4. $\mathcal{P}$ commits to all d_i as $[\![d_i]\!]$, and proves $p[\![d_i]\!] = v_i - m_i - u_i[\![s]\!]$ using Π_{LIN}.
5. $\mathcal{P}$ uses protocol Π_{A} to prove that all $\|d_i\|_2$ are bounded by $B_{\text{A}} \leq \sqrt{2vN}\sigma_{\text{A}}$.
6. $\mathcal{P}$ outputs messages $\{m_i\}_{i=1}^{\tau}$, commitments $\{[\![d_i]\!]\}_{i=1}^{\tau}$ and proofs $\{\pi_{L_i}\}_{i=1}^{\tau}, \pi_{\text{A}}$.

A verifier $\mathcal{V}$ runs the verification protocol Π_{DecV} which checks that all proofs $\{\pi_{L_i}\}_{i=1}^{\tau}$ and π_{A} are valid with respect to (a, b), $\{(u_i, v_i)\}_{i=1}^{\tau}$ and $\{m_i\}_{i=1}^{\tau}$.

Theorem 1. *The verifiable decryption protocol Π_{Dec} is a complete, sound and zero-knowledge proof protocol in the ROM for relation R_{Dec} when $B_{\text{A}} < q/(4p)$.*

Proof. We argue each of the properties as following:

Completeness. It follows directly that Π_{Dec} is complete if the encryption scheme is correct, which is the case when $\|v - su\| < q/2$, and the protocols Π_{LIN} and Π_{A} are complete. Hence, we only need to make sure that $\|v - su\| < q/2$. The protocol Π_{A} guarantees that the noise is bounded as $\|d_i\|_2 \leq 2B_{\text{A}}$. It follows that if $B_{\text{A}} < q/(4p)$ then $\|d_i\|_\infty < q/2p$, and the decryption is correct.

Special soundness. The soundness of the protocol follows directly from the underlying zero-knowledge protocols Π_{LIN} and Π_{A}. With the use of rewinding we can either extract the secret key s or the noise d_i (which reveals the secret key) or some short vectors breaking the SKS^2 problem for the given parameters.

Honest-verifier zero-knowledge. The zero-knowledge property follows directly from the underlying zero-knowledge protocols Π_{LIN} and Π_{A}, which are both honest-verifier zero-knowledge. Hence, with input messages $m_1, \ldots, m_\tau$ we can simulate the decryption proof by sampling uniformly random values d_i, committing to them as $[\![d_i]\!]$ and then simulating all the proofs π_{L_i} and π_{A} subsequently. □

4 Performance

4.1 Parameters and Size

From the verifiable decryption protocol in Section 3 we get that the statement consists of τ ciphertexts (u_i, v_i) and messages m_i. Each element u_i and v_i are uniformly elements in R_q of size $N \log_2 q$ bits each. The messages are elements in R_q with coordinates modulo p, and hence, are of size $N \log_2 p$ bits. Each proof π_L are of size $2(k-n)N \log_2(6\sigma_{\text{C}})$ bits, for $\sigma_{\text{C}} = 11\nu\beta_\infty\sqrt{kN}$, and the proof π_{A} is of size $(k+1)\hat{n}N \log_2(6\sigma_{\text{A}})$ bits. However, the norm bound B_{A} depends on the number of equations being proved at once, and hence, if τ is large it is beneficial to prove smaller batches, e.g., of size N, instead of all equations at once.

As a concrete example, we set $p = 2$, $\beta_\infty = 1$ and let $\mathtt{D}$ be the ternary distribution over R_q. It then follows that, for honestly generated ciphertexts,

p	q	N	β_∞	M	k	n	$\hat{n}$	ν	σ_{C}	B_{C}	σ_{A}	B_{A}
2	$\approx 2^{44}$	2048	1	3	3	1	130	36	$\approx 2^{15.9}$	$\approx 2^{22.4}$	$\approx 2^{28}$	$\approx 2^{40.5}$

Table 1. Example parameters for the verifiable decryption protocol with more than 128 bits of security against quantum adversaries ensuring correct decryption for honestly generated ciphertexts. Rejection sampling success probability is set to be $\approx 1/3$.

$\|v - us\|_\infty \leq p(2N+1)$. Furthermore, we get the following bound for $\|d_i\|_\infty$:

$$\begin{aligned}\|d_i\|_\infty &\leq 2B_{\mathrm{A}} = \sqrt{8N}\sigma_{\mathrm{A}} \leq \sqrt{8N} \cdot 0.675\|\boldsymbol{S}'\boldsymbol{C}'\|_2 \leq 2\sqrt{N} \cdot \|\boldsymbol{S}'\|_1 \|\boldsymbol{C}'\|_\infty \\ &\leq 2\sqrt{N} \cdot (\beta_\infty kN + p(2N+1)N)\tau.\end{aligned}$$

Thus, setting $k = 3, n = 1, \hat{n} = 130, \nu = 36$ and $\tau = N = 2048$ gives us $\|d_i\|_\infty \leq 2^{41.5}$, and we can safely set $q \approx 2^{44}$ to get correctness. We claim more than 128 bits security against a quantum adversary for these parameters using the LWE estimator by Albrecht *et al.* [APS15] with the **BKZ.qsieve** cost model. A smaller N results in smaller noise, but the size of q would give lower security.

Message m_i	Ciphertext (u_i, v_i)	Commitment $[\![d_i]\!]$	Proof π_{L_i}	Proof π_{A}	Proof π_{DEC}
0.256 KB	22.6 KB	22.6 KB	19 KB	2τ KB	43.6τ KB

Table 2. Sizes for parameters $p = 2, q \approx 2^{44}$ and $N = 2048$ computing proof $\pi_{\mathrm{DEC}} = (\{[\![d_i]\!], \pi_{L_i}\}_{i=1}^{\tau}, \pi_{\mathrm{A}})$, where shortness proofs π_{A} is amortized over batches of size 2048.

4.2 Implementation and Timings

We provide a proof-of-concept implementation of our protocol in C++ using the NTL-library [Sho21]. The implementation was benchmarked on an Intel Core i5 running at 2.3GHz with 16 GB RAM. The timings are given in Table 3. The implementation is very simple, consists of a total of 350 lines of code, and is available online[⋆]. A comparison of NTL to NFLlib [ABG+16] indicates that an optimized implementation could provide speedup by an order of magnitude.

Noise $[\![d_i]\!]$	Proof Π_{LIN}	Verification Π_{LINV}	Proof Π_{A}	Verification Π_{AV}	Proof π_{DEC}
5τ ms	47τ ms	12τ ms	24τ ms	12τ ms	76τ ms

Table 3. Amortized time per instance over $\tau = 2048$ ciphertexts.

4.3 Comparison

We compare to the verifiable decryption protocols by Lyubashevsky *et al.* [LNS21] and Gjøsteen *et al.* [GHM+22]. As noted by [GHM+22, Section 8], the protocol by Boschini *et al.* [BCOS20] give proof sizes of approximate 90 KB, which is roughly twice the size of π_{DEC}. Furthermore, the run time is several minutes per ciphertext, which would deem it unusable for larger sets of ciphertexts.

[⋆] github.com/tjesi/verifiable-decryption-BGV.

Comparison to Lyubashevsky *et al.* (PKC 2021). They give a verifiable decryption protocol for the Kyber encapsulation scheme for a ring of dimension $N = 256$ and modulus $q = 3329$ with secret and noise values bounded by $\beta_\infty = 2$. The proof of correct decryption is of size 43.6 KB. We observe that our proof is of exactly the same size but with a plaintext space of 2048 bits instead of only 256 bits. We expect our proof size to be smaller than theirs for ciphertexts encoding larger messages, but note that they can provide efficient proofs for single ciphertexts for small moduli while our protocol is only efficient in the amortized setting for ciphertext moduli at least 44 bits. Furthermore, our protocol is much simpler, as [LNS21] make use of partially splitting rings and automorphisms by combining proofs of multiplication and range proofs – making the protocol difficult to implement in practice. They do not provide timings.

Comparison to Gjøsteen *et al.* (ACISP 2022). They give a verifiable decryption protocol $\Pi_{\mathtt{ZKPCD}}$ for the BGV encryption scheme. However, because of their noise drowning techniques, they are forced to use a moduli of at least $q \approx 2^{55}$. Their proof size is also depending on the soundness parameter λ, giving a proof of size 14λ KB per ciphertext. For an interactive protocol with $\lambda = 10$ they get a proof of size $3.2\times$ larger than our proof, and for a non-interactive protocol with $\lambda = 128$ their proof size is $41\times$ larger than ours. They have implemented their protocol, and give a cost of at least 4λ ms per ciphertext using NTL, which is similar to our protocol for $\lambda = 19$ and otherwise slower.

They also sketch a protocol $\Pi_{\mathtt{DistDec}}$ [GHM+22, Section 8], requiring $q \approx 2^{78}$ and $N = 4096$. This protocol gives a proof of size ≈ 363 KB per ciphertext, a factor 8 larger than our proof. They do not provide timings for this protocol.

References

ABG+16. Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. NFLlib: NTT-based fast lattice library. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 341–356. Springer, Heidelberg, February / March 2016.

ABG+21. Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde, and Thor Tunge. Lattice-based proof of shuffle and applications to electronic voting. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 227–251. Springer, Heidelberg, May 2021.

Adi08. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008*, pages 335–348. USENIX Association, July / August 2008.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 2015.

BBC+18. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 669–699. Springer, Heidelberg, August 2018.

BCOS20. Cecilia Boschini, Jan Camenisch, Max Ovsiankin, and Nicholas Spooner. Efficient post-quantum SNARKs for RSIS and RLWE and their applications to privacy. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum*

Cryptography - 11th International Conference, PQCrypto 2020, pages 247–267. Springer, Heidelberg, 2020.

BDL$^+$18. Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 368–385. Springer, Heidelberg, September 2018.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

CMM19. Núria Costa, Ramiro Martínez, and Paz Morillo. Lattice-based proof of a shuffle. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *FC 2019 Workshops*, volume 11599 of *LNCS*, pages 330–346. Springer, Heidelberg, February 2019.

CS03. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, August 2003.

CWF13. Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In Samuel T. King, editor, *USENIX Security 2013*, pages 147–162. USENIX Association, August 2013.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

FWK21. Valeh Farzaliyev, Jan Willemson, and Jaan Kristjan Kaasik. Improved lattice-based mix-nets for electronic voting. Cryptology ePrint Archive, Report 2021/1499, 2021. `https://ia.cr/2021/1499`.

GHM$^+$22. Kristian Gjøsteen, Thomas Haines, Johannes Müller, Peter Rønne, and Tjerand Silde. Verifiable decryption in the head. ACISP, 2022. `https://eprint.iacr.org/2021/558.pdf`.

HM20. Thomas Haines and Johannes Müller. SoK: Techniques for verifiable mix nets. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 49–64. IEEE Computer Society Press, 2020.

LM06. Vadim Lyubashevsky and Daniele Micciancio. Generalized compact Knapsacks are collision resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 144–155. Springer, Heidelberg, July 2006.

LNS21. Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 215–241. Springer, Heidelberg, May 2021.

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

LW18. Fucai Luo and Kunpeng Wang. Verifiable decryption for fully homomorphic encryption. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *ISC 2018*, volume 11060 of *LNCS*, pages 347–365. Springer, Heidelberg, September 2018.

Sho21. Victor Shoup. Ntl: A library for doing number theory, 2021. `https://libntl.org/index.html`.

ISBN 978-82-326-6571-6 (printed ver.)
ISBN 978-82-326-5212-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)