KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Secure and Efficient Computing on Private Data

Eleftheria Makri

Supervisors:
Prof. dr. ir. Bart Preneel
Prof. dr. ir. Frederik Vercauteren

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

July 2021

Secure and Efficient Computing on Private Data

Eleftheria MAKRI

Examination committee:
Prof. dr. ir. Paul Sas, chair
Prof. dr. ir. Bart Preneel, supervisor
Prof. dr. ir. Frederik Vercauteren, supervisor
Prof. dr. Nigel P. Smart
Prof. dr. Yves Moreau
Dr. Joppe W. Bos (NXP Belgium)
Prof. dr. Carmit Hazay
(Bar-Ilan University, Israel)
Prof. dr. Peter Scholl
(Aarhus University, Denmark)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

July 2021

Uitgegeven in eigen beheer, Eleftheria Makri, Kasteelpark Arenberg 10 box 2452, B-3001 Leuven (Belgium)

Acknowledgements

Almost 4,5 years ago a wonderful journey started, my trip with COSIC; a real roller coaster of emotions and experiences. Writing the last words to conclude this journey, and despite my eagerness to finish my PhD, I cannot say I am happy; the taste is more a bitter-sweet combination. This journey, and the completion thereof, would never be possible without the support of some people, and I can mention but a few of them.

First and foremost, I would like to thank my promotors, Bart Preneel, and Fre Vercauteren. You trusted me (for no particular reason) to start a PhD in the group, while having another part-time job, and living in another country. You gave me the freedom to do research on the topics of my choice, while at the same time always being there to mentor me. Fre, I have learned so much from you, on content, on doing research, on being critical, and beyond. I am truly grateful for your support!

I am very grateful to my supervisory committee, Joppe Bos, Yves Moreau, and Nigel Smart for keeping track of my PhD progress, and providing me with valuable feedback throughout these years. Special thanks to Nigel, for I have learned so much from you, as well. You have been, in a way, also sort of a co-supervisor for me. I would like to extend my gratitude to the external members of my examination committee, Carmit Hazay, and Peter Scholl. Thank you for all the work you put into reading and assessing my work, and for your constructive feedback, which helped me make this thesis better. I also thank Prof. Paul Sas for chairing my defenses.

Very special thanks go to my co-authors: Amin Ardeshirdavani, Charlotte Bonte, Cyprien Delpech de Saint Guillem, Yves Moreau, Dragos Rotaru, Jaak Simm, Nigel Smart, Titouan Tanguy, Fre Vercauteren, Sameer Wagh, and Tim Wood. Without you, completing this thesis would have been impossible. I am forever indepted to you for all you have taught me. For the exciting research moments (including breaking what we considered a great result for a day, sometimes!), for

the fun times, for the stressful times (editing 74 seconds before the submission deadline!), for your trust and support, and most importantly –for many of you– for your friendship. I admire your spirit and intelligence, and I wish to work with you again in the future.

COSIC is not a group, it is a family. Many thanks to all of my colleagues (or by now friends), for the wonderful, unforgettable 4,5 years we spent together. I have enjoyed beyond work, our game nights, the few Friday-beers I could join, the lunch talks, and insightful conversations during the coffee breaks, halloween parties, Christmas parties, seeing many of you becoming doctors, and celebrating with you, and many more events that I am skipping. Many of you have a special place in my heart, but I made a very special friend for life in the group, Charlotte, who is simply an amazing person, a great listener, an intelligent researcher, a helpful collaborator, and a person to have real fun with. As every family has a mom, COSIC has Péla. Thank you for your genuine interest in me and my progress (and since recently my puppy). For always supporting me no matter what the request, and for solving all my administration problems. I would also like to thank Dana, Saartje, Wim, and Elsy for all the administrative support, and for taking care of the publicity of any event I was participating, even when the announcement arrived last moment.

I am also very grateful to my employer, Saxion University of Applied Sciences, for trusting me with such a challenging endeavor, and for funding my PhD.

Last but not least, I would like to thank my family for always supporting me in whatever I undertake (regardless of whether they think it is crazy, or not). Despite the thousands of kilometers between us, you have always been next to me! I thank you for that immensly! I love you very much! Elmer, thank you so much for believing in me so strongly, for always giving me a hand to stand up again when I fall, for loving me and letting me love you, for being a great friend, next to being my partner in crime (I mean in life)!

Eleftheria Makri
Leuven, July 2021

Abstract

Multiparty Computation (MPC) is a cryptographic primitive that allows a set of mutually distrusting parties to compute a function over their private inputs, without revealing to each other or to outside observers these inputs. In fact, MPC allows one or more of the protocol participants to obtain the output of the function to be computed, while learning no more information about other participants' inputs, beyond what can be infered from the protocol's output. Due to its rich functionality, allowing the computation of virtually any function on private data, MPC is a largely researched cryptographic primitive.

The study of MPC dates back to the 80's, when Yao introduced the well-known *Millionaires' Problem*, and proposed a solution for the problem. Yao's millionaires' problem is to determine who is richer out of two parties, in a manner that leaks no information about the amount of assets of one party to the other. Despite the numerous theoretical and practical research results in the field of MPC, the area is still actively researched as the goal is to design solutions that are efficient for practical use, while satisfying strong provable security guarantees, and have a rich functionality at the same time. MPC research today aims at combating the inherent tradeoffs among efficiency, security, and expressiveness of the computable functions, and combine the best of all three worlds either for a specific cryptographic primitive, or for a particular application scenario.

In this thesis, we aim at improving as many of the three aforementioned aspects as possible to construct both application-scenario-tailored solutions, and improved MPC building blocks.

Firstly, we demonstrate that large-scale secure computation is practical, by designing a solution for privacy-preserving Genome-Wide Association Studies. In this setting, we compare the performance and security of two secure computation methods: MPC, and Homomorphic Encryption (HE), and confirm that MPC both outperforms HE, and can provide security in a more stringent security model.

Secondly, we design a private image classification protocol, which outperforms the state-of-the-art in this sub-area at the time of publication. We show how to leverage knowledge from complementary research fields, in this case from the machine learning community, to boost the efficiency of our secure designs, while also increasing the classification accuracy compared to previous secure solutions.

Thirdly, we construct a protocol to securely generate RSA moduli. Our protocol is more communication-efficient than the state-of-the-art, and it is comprised by building-blocks (i.e., other sub-protocols), which we designed, that are of independent interest.

In addition, we show how to garble multiparty arithmetic circuits with full-threshold active security, and improve on the communication efficiency of a specific garbled gate, namely a selector gate.

Finally, we devise a generic secure comparison protocol, outperforming the state-of-the-art, while having the potential of offering higher security guarantees, should the underlying implementation platform allow that. Secure comparisons are a fundamental MPC building block that is both necessary for the realization of larger tasks (e.g., online auctions), and challenging in terms of efficiency.

Beknopte samenvatting

Een beveiligde berekening met meerdere partijen (Engels: Multiparty computation of MPC) is een cryptografisch primitief dat toelaat om een functie te berekenen op private data van meerdere elkaar wantrouwende partijen zonder deze invoerdata beschikbaar te maken voor andere partijen. Meer specifiek maakt MPC het mogelijk dat een of meerdere van de partijen betrokken in de berekening de uitkomst van de beveiligde berekening in handen krijgen, zonder ook maar enige informatie te krijgen over de invoerdata van andere partijen buiten wat afgeleid kan worden uit het resultaat en zijn eigen invoerdata. Omwille van de talrijke mogelijkheden die MPC biedt, aangezien het vrijwel toelaat om elke mogelijke berekening op een gegevens-beschermende manier uit te voeren, is het een populair research topic.

MPC gerelateerd onderzoek startte in de jaren '80, toen Yao het inmiddels bekende *Miljonairs Probleem* bedacht en bovendien een oplossing voor dit probleem aanbood. In het Miljonairs Probleem van Yao moet bepaald worden welke van twee partijen rijker is, zonder dat de partijen iets te weten komen over de omvang van de bezittingen en rijkdommen van de ander. Ondanks de enorme hoeveelheid theoretische en praktische onderzoeksresultaten, is MPC nog steeds een actief onderzoeksveld met als doel het ontwikkelen van efficiënte oplossingen met sterke veiligheidsgaranties en brede functionaliteiten. Huidig onderzoek in MPC zoekt naar de balans tussen efficiëntie, veiligheid en de juiste vormgeving van de berekeningen om de beste oplossing te creëren voor een specifieke primitieve functie of bepaald toepassingsscenario.

Dit proefschrift is gericht op het verbeteren van de drie genoemde aspecten om zo toepassingsspecifieke oplossingen te ontwikkelen en primitieven voor MPC te perfectioneren.

In een eerste werk tonen we aan de hand van een genoom-brede associatie analyse aan dat het praktisch haalbaar is om beveiligde berekeningen op grote schaal uit te voeren. In dit toepassingsscenario vergelijken we de prestaties en

beveiliging van twee methoden voor het uitvoeren van beveiligde berekeningen vergeleken: MPC en Homomorfe Encryptie (HE). Dit toont aan dat MPC HE overtreft zowel op vlak van efficiëntie als op vlak van de veiligheidsgaranties.

Vervolgens hebben we een protocol ontwikkeld om op een privacy-beschermende manier afbeeldingen te categoriseren, waarbij ons design op moment van publicatie sneller was dan de State of the Art. We combineren de kennis uit een ander vakgebied, in dit geval machine learning, om de efficiëntie van onze privacy-beschermende oplossing te verbeteren. Onze oplossing verbetert bovendien de nauwkeurigheid van de classificatie ten opzichte van eerdere privacy-beschermende oplossingen.

Ook hebben we een protocol gemaakt waarmee we op een veilige manier RSA moduli kunnen genereren. Ons protocol biedt efficiëntere communicatie dan de State of the Art en bestaat uit, door ons ontworpen, bouwstenen (subprotocollen) die ook zelfstandig bruikbaar zijn.

Daarnaast tonen we hoe we arithmetische circuits met meerdere partijen kunnen obfusceren in de context van actieve beveiliging waarbij alle partijen op één na kwaadaardig zijn. We verbeteren de efficiëntie van de communicatie van een specifieke geobfusceerde logische poort, namelijk de multiplexer.

Tot slot hebben we een universeel vergelijkingsprotocol ontwikkeld dat sneller is dan de State of the Art en, wanneer de onderliggende implementatie dat toestaat, ook veiliger is. Privacy-beschermende vergelijkingen zijn fundamenteel voor het realiseren van grotere taken (bijvoorbeeld online veilingen) maar het efficiënt realiseren van deze vergelijkingen is een uitdaging in MPC.

Contents

List of Figures

Part I

Secure and Efficient Computing on Private Data

Chapter 1

Introduction

We live in the era of information, and it is becoming more and more common, not only for businesses, but also for leisure activities, healthcare, and daily personal activities to be information-driven, or at least information-based. The interconnected person of today's developed world cannot work, perform day-to-day operations, or even imagine their lifestyle without the use of information and information technology. All this information, which was initially stored and processed locally under the user's supervision, is now being processed by distributed systems, and online applications. The advent of Cloud Computing, the trend of Big Data, as well as the Internet of Things, which aims at the integration of the physical world with the computer-based systems, necessitate the information collection and processing to occur globally.

When appropriately processed (e.g., by using data mining techniques), all these massive amounts of data can be of great benefit to society. Machine learning, and other applications of artificial intelligence, offer excellent opportunities. Concrete application scenarios include ambient intelligence settings, such as smart homes and smart cities, where data collection can improve efficiency and accuracy of the present infrastructure, resulting in both economic and environmental benefits [28]. Advances in the field of medicine, such as predicting epidemics, or devising new drugs, can be attributed to Big Data [72]. The emerging home (self-)healthcare applications that appear to be the "care of the future", given our ageing population, and the predicted shortage of medical doctors, present themselves as further interesting scenarios, where Big Data is indispensable [74]. Furthermore, supporting the efforts of public emergency services with real-time geolocation information can enable rapid, effective, and efficient rescue operations [69]. The emergence of all sorts of autonomous

systems, including autonomous driving applications, can fundamentally change the way transportation is being implemented nowadays, and it is facilitated by the collection of detailed sensor and map data [20]. In addition, current everyday activities involve the dissemination of information amongst others for leisure activities, shopping, professional and social networking that take place online. Due to the vast amount of data that is required so as to facilitate these online applications, users are often prompted to share personal information, in order to receive more personalized recommendations for products, and targeted advertisements [62].

Despite the profound benefits that information dissemination bares in the context of the aforementioned applications, it comes at the cost of privacy, which is a fundamental human right. The end users of online applications, such as the ones facilitating the previously mentioned scenarios, run the risk of having their privacy compromised, while disseminating their personal information. People are still unaware of the implications of privacy breaches, often using the cliché "I do not care about privacy, because I have nothing to hide". Clearly, this demonstrates that many users do not understand what privacy is about, and how related breaches can be used against them. Privacy breaches can be accidental, intentional, or simply the result of user's negligence. Privacy breaches go beyond the discovery of user's personal preferences, or the discovery of user's passwords for particular websites. When users claim having nothing to hide, they have not considered scenarios, where their sensitive health information gets released, resulting in them being unemployable (due to health conditions that became public); or resulting in an insurance company not accepting them as clients, or increasing the insurance fees.

When users regularly release their location data, to facilitate ambient intelligence applications, such as autonomous HVAC (Heating, Ventilation, and Air-Conditioning) systems, or as part of their daily exercises and participation in a social sport network, they essentially release patterns of their behavior that can be used against them. It has been shown that combination of spatial and temporal information shared by the users of online social sports networks can facilitate physical crime [93]. Predicting where a user lives, when they go out to exercise, and for how long, allows criminals to plan and execute burglaries, or to otherwise offend the users. Although it seems to be the most innocent of the applications collecting personal information, in terms of privacy violations, a user's shopping behavior can reveal significant information about themselves. Companies can analyze buying behavior, and even predict certain personal circumstances in one's life before the persons in question themselves. By predicting these circumstances accurately and timely, companies can deploy very effective marketing strategies; but all this at the cost of privacy. For example, a teenager in the U.S.A. was predicted by the Target store to be

pregnant, before she knew it herself [41]. This demonstrates how high the impact of collection of personal information can be.

Even when users trust the respective service provider with their personal information, data breaches cannot be excluded. Suppose that we trust the emergency services of the city where we live, to collect and process our location information throughout the day, so as to assist in the progress of potential rescue operations. We trust that the emergency services will treat our information as confidential, while in transit, but a hacking attempt can compromise the confidentiality of our data, and eventually our privacy, at the processing site of the emergency services itself. Thus, trust in the service provider cannot guarantee privacy preservation. These are not fictional scenarios, but rather an everyday phenomenon. Prevalent examples of data breaches and information leaks include one of the recent LinkedIn breach cases, revealing information about 117 million accounts [19], the Ashley Madison breach, exposing married people's extramarital affairs [58], and the Cambridge Analytica Scandal in 2018 [97].

Based on the previous discussion, the following question is raised: "How to allow the collection and purposeful processing of private data, without compromising individual privacy?". To address this research question, several approaches have been examined; one of them is data anonymization. Data anonymization techniques have been shown to be easily circumvented, if the data allows other kinds of correlations [94]. More recent results by Narayanan and Shmatikov [80], remove the "a priori knowledge" requirement, rendering de-anonymization even more easily deployable. Another approach is differential privacy [40, 43]. Unlike anonymization approaches, differential privacy is robust against adversaries aiming at recovering personally identifiable information, even when these possess auxiliary information [42], and it offers provable privacy guarantees. However, differential privacy applies primarily to statistical databases, and the noise introduced to preserve privacy directly affects the accuracy of the query responses.

In cases where the data processing necessary for an application goes beyond a statistical database, or in cases where the accuracy of the private computations is important, the aforementioned techniques are insufficient. This thesis focuses on cryptographic techniques for secure computation, which are deployable not only when individual privacy –in terms of personally identifiable information– is concerned, but also when the confidentiality of the underlying data involved in the computation needs to be protected. Example applications in this setting include e-voting [31], online auctions [11], the more recent, also commercialized, applications on cryptographic key management [95, 89], and the renowned millionaires' problem [98], Yao's seminal work that initiated the study of secure computation. Yao's millionaires' problem reads as follows: *"Two millionaires*

wish to know who is richer; however, they do not want to find out inadvertently any additional information about each other's wealth. How can they carry out such a conversation?" [98]. Generalizing the abovementioned problem, we come to the overarching research question that we aim to address in this thesis: "How can n mutually distrusting parties compute a predetermined function $f(\cdot)$ over their private inputs $x_1, \ldots, x_n$, in such a way that at the end of the computation each party learns the output of the computation $f(x_1, \ldots, x_n)$ and nothing more." Specifically, we devise protocols to tackle this research challenge, while being efficient, and provably secure.

1.1 Motivation for this work

Theoretically, an affirmative answer on whether we can achieve secure computation has been already given in the 80's, together with constructions on how to realize it [98, 99, 5, 87, 86]. Since then, a lot of research has been done in the field, but the proposed solutions are commonly subjected to some kind of compromise. Some approaches, although provably secure, offer only limited functionality (e.g., they only support secure addition on the private inputs), which in certain application scenarios is insufficient to realize the desired processing. Other solutions involve the execution of interactive protocols among several parties, who potentially contribute their private inputs, and require these parties to be available online, in a synchronous manner, and be actively involved themselves to complete the computations. In addition, there are solutions that may offer the necessary functionality to complete the tasks at hand, but they are so inefficient that they cannot be deployed on real-life applications. The efficiency requirement itself is further divided into computational, communication, and memory efficiency, as well as throughput of certain operations, especially when computations can be parallelized. Lack of efficiency can render secure computation protocols inapplicable in practice, as their costs –be it computational, communication, or memory costs– can become impractical to be handled by current computing systems. Lastly, secure computation solutions are proven secure in different security models, under different security assumptions, where generally the weaker the security offered by a secure computation protocol, the more efficient, or functional the solution is. All in all, there are tradeoffs attributed to each secure computation approach that can be appropriately adjusted to serve specific application scenarios.

The three main subfields of cryptography that aim at providing solutions to the problem of secure computation are homomorphic encryption (HE), Secure Multiparty Computation (MPC) based on Secret Sharing, and MPC based on Garbled Circuits (GC). We elaborate on all three mentioned subfields in

Chapter 2, but this thesis focuses on Secure Multiparty Computation, both based on Secret Sharing, and based on Garbled Circuits. Concretely, we deploy and devise general n-party MPC protocols (while there are also two-party protocols, enjoying special properties) that are secure in the most stringent security model, the active security model and the dishonest majority setting. In this endeavor, there are two approaches one may take: (1) consider a specific application scenario, and tailor the secure computation solution perfectly to the scenario, so as to achieve the best tradeoff in the particular setting; (2) improve a particular secure computation primitive that is of general nature, and applies to many different application scenarios. In this thesis, we have taken both approaches, with the common underlying goal of providing solutions that are practical in terms of efficiency, competing favorably with the previous state-of-the-art.

1.2 Summary of Contributions

Taking the first approach on the deployment of secure computation, tailoring the protocols to a particular application scenario, this thesis makes the following contributions:

We show on a specific use-case, namely Genome-Wide Association Studies (GWAS), that large-scale secure computation is practical. We study the GWAS problem in depth, and discover privacy-challenges that fall beyond the goals of secure computation. Secure computation protects the private inputs of the protocol, and allows the protocol participants and outside observers to learn nothing more than the output of the protocol, and whatever can be inferred from that. In the GWAS application, even what can be inferred from the output of a secure χ^2-statistic computation can reveal information about individual research subjects [59]. To tackle this, we tailor our solution, so as to not reveal the actual χ^2-statistic value, but rather a flag on whether the value is below or above a predetermined threshold, which indicates statistical significance. We compare the efficiency and security of two secure computation methods, HE and MPC, on a real-life application with appropriate sample sizes, for a research task that scales to millions of individuals contributing to the samples. We propose a masking technique, which allows to perform more efficiently the required secret comparison that stems from the privacy requirement to not reveal the χ^2-statistic. The deployment of this comparison strategy improves the efficiency of the HE-based solution.

We devise an efficient private image classification system, called EPIC, based on Support Vector Machine (SVM) learning, and secure

against active adversaries. Outsourcing an image classification task raises privacy concerns, both from the image provider's perspective, who wishes to keep their images confidential, and from the classification algorithm provider's perspective, who wishes to protect the intellectual property of their classifier. EPIC addresses the privacy concerns of both of these parties, and allows for complete outsourcing of the image classification task to a third independent party. Although conceptually simple, based on SVM learning, EPIC is the first actively secure solution for image classification, it does not leak any information about the private images, nor the classifier, and yet it is faster and more accurate than its predecessors, which employ more complex machine learning techniques. We show how to increase the accuracy of a privacy-preserving approach without necessarily involving computations that are inefficient to perform in a secure manner, by deploying data-independent feature extraction methods that can be performed prior to the privacy-preserving computations without affecting privacy. Our implementation and experiments on realistic datasets demonstrate that our approach is practical, both in terms of accuracy, and in terms of efficiency.

We design a MPC protocol that securely generates an RSA modulus. Generating an RSA modulus securely is a challenging task that finds several applications nowadays. We design a generic protocol for the task, supporting any underlying MPC technology, as long as it is based on linear secret sharing. Our protocol is based on a distributed sieving technique, and leverages the Chinese Remainder Theorem (CRT) for the representation of the shares to increase efficiency. In our effort to design a novel protocol, which is efficient, we design a sub-protocol that converts an additive sharing over a ring to an additive sharing over the integers. This sub-protocol can be of independent interest. We also performed a cost analysis of our approach, and compared it with the state-of-the-art in this area [24]. Our protocol improves the communication cost of the previously proposed solution. Particularly, in the case of malicious security, candidate primes of 2048 bits, and 2 parties, we show a communication cost improved by 37 times.

Looking at general purpose, rather than application-specific secure computation primitives to be improved, this thesis makes the following contributions:

We show how to garble arithmetic circuits with full-threshold active security in the general multiparty setting. Firstly, we extend the work of Ben-Efraim [7] from the semi-honest security setting, to support malicious security against up to $n-1$ (out of the total n) corrupted parties. Like Ben-Efraim's protocol, we show how to allow interfacing between Boolean and arithmetic circuit gates. Lastly, we provide a new construction for a particular garbled gate, namely a selector gate (i.e., a gate that given two arithmetic inputs and one binary input, outputs one of the two arithmetic values, based

on the value of the binary input). The newly proposed selector gate reduces the communication cost to almost half of the cost of its predecessor.

We devise an efficient comparison protocol for secure multiparty computation. Our comparison protocol is based on the commutative nature of addition over rings and fields, and it is information theoretically secure when the underlying MPC protocol operates over $\mathbb{Z}_{2^k}$. Unlike prior work, our comparison protocol does not require the computation to take place in a statistically larger dataspace to ensure security. The consequence of that is that we can perform both the comparison operation, as well as all adjacent computations on smaller datatypes, because we do not need a larger space to accommodate for the statistical security parameter. Our design results in a straightforward protocol, which we implemented in MP-SPDZ [63, 36]. We experimentally show that in most adversarial settings our protocol efficiency compares favorably with the previously fastest comparison protocol [45].

1.3 Organization

In the first part of this thesis, following the Introduction (Chapter 1), we elaborate in Chapter 2 on the three main secure computation methods. In Chapter 3 the fundamental cryptographic primitives serving as building blocks throughout this work are explained. Chapter 4 discusses the different security models and assumptions in the area of secure computation, and MPC in particular. In Chapter 5 we present the neccessary ingredients to construct MPC protocols in the *preprocessing model*, which is the model assumed in Part II of this thesis, while Chapter 6 summarizes our conclusions, and lists some interesting open questions, which may lead to future work.

The second part of this thesis is comprised by the five peer-reviewed, or under submission papers (in chronological order of publication), which are the main results of this work, namely:

1. Two methods to perform Genome-Wide Association Studies (GWAS) and in particular securely computing the χ^2-statistic, where we show that such studies, scaling to millions of subjects, are practical to do in a privacy-preserving manner [13]. We take two approaches on addressing this challenge: a somewhat homomorphic encryption approach, and a MPC computation approach, and compare how these perform in terms of security and efficiency.

2. An efficient private image classification protocol, EPIC, which is a MPC protocol that builds upon transfer learning techniques to achieve efficiency

significantly improved compared to the state-of-the-art, while maintaining the image classification accuracy [75].

3. A secure comparison protocol based on the commutative nature of addition over rings and fields [76], with increased efficiency in most adversarial settings, over the most recent secure comparison protocol [45].

4. A multiparty arithmetic garbling protocol, which is proven secure in the full-threshold active security model, and it is the first of its kind in terms of security [77]. At the same time we show how to reduce to almost half the asymptotic communication cost of a particular garbled gate, namely a selector gate, compared to the previously proposed solution [7].

5. A protocol to securely generate a biprime, based on a distributed sieving technique and leveraging the CRT representation [38], which is shown to improve the communication cost over the best previous work [24] by up to a factor of 37.

Chapter 2

Secure Computation Methods

This chapter elaborates on the three main secure computation methods, namely homomorphic encryption, multiparty computation based on secret sharing, and multiparty computation based on garbled circuits.

2.1 Homomorphic Encryption

Encryption allows the encoding of (sensitive, or private) data, into a different form, which does not reveal any information about the original data. The encryption operation can only be inverted by authorized parties to retrieve the original data. Homomorphic encryption (HE) is a form of encryption allowing computations to be performed on private data, where the eventual private result of the computation, when decrypted, matches the result of a possibly different operation performed on the original data. The original (private) data contributed to the computation is called *plaintext* data, following the same terminology as in traditional encryption. This data gets encrypted, following the encryption algorithm, with input the plaintext, and the public or private *key*. This process results in the *ciphertext*. With HE the ciphertext can be processed according to the computations allowed by the HE scheme. The result of this processing is still in encrypted form, and the only entities able to decrypt this are the ones in possesion of the private key of the cryptosystem. To do so, they use the ciphertext and the private key as inputs to the decryption algorithm. Once decrypted, the resulting plaintext is the result of the performed computations, as if these had been performed on the plaintext data.

The concept of privacy homomorphisms [86] was formalized in the same year as the first public key encryption scheme was proposed, namely the RSA scheme [87], which itself enjoys the property of being homomorphic (i.e., allowing certain computations to be performed on the ciphertext). More precisely, RSA is multiplicatively homomorphic in that multiplication of two encrypted integer numbers, results in an encryption of the result of the multiplication on the original (unencrypted) integers. The homomorphic property of RSA can be employed only when the scheme is being used unpadded, which is not recommended, as unpadded RSA is not secure.

Following RSA, Goldwasser, and Micali [56] invented another public key cryptosystem, with the following homomorphic property: multiplication of two encrypted bits, results in an encryption of the result of an exclusive OR (XOR) operation on the original plaintext bits. Later, ElGamal [44] proposed the second multiplicatively homomorphic encryption scheme (after RSA). The first additively homomorphic encryption scheme (operating on encrypted *integers* instead of encrypted *bits* like the scheme of Goldwasser, and Micali [56]) was presented by Benaloh [8]. With Benaloh's scheme [8] the homomorphic property is that the product of two encrypted integers, results in an encryption of the result of an addition operation on the original plaintext integers. The famous encryption scheme of Paillier [82] enjoys the same homomorphic property as the one of Benaloh [8]. Paillier's cryptosystem has been widely deployed, also in combination with secure multiparty computation techniques, to address some of the important research challenges we discuss in Chapter 1. Regev's encryption scheme [85], which is based on the Learning With Errors (LWE) problem, also enjoys an additively homomorphic property.

The encryption schemes discussed so far in the first part of this section are *Partially Homomorphic Encryption* (PHE) schemes. This means that with these encryption schemes, we can perform either multiplication or addition operations on the encrypted data, but not both at the same version of the encrypted data. Note that all operations on integers boil down to addition, and multiplication. If we are able to perform both of these operations on the same version of the encrypted data, we can adequately address the challenges of privacy enhancing technologies introduced in Chapter 1. When working with Boolean circuits, a homomorphic encryption scheme that allows unlimited multiplications and additions to be performed on the encrypted data, allows us to represent encrypted NAND gates, which require one addition and one multiplication operation to be performed on the inputs. We remark that NAND gates alone provide functional completeness, which means that in the above scenario, we can evaluate any circuit of any depth on the encrypted inputs. An encryption scheme satisfying the above properties is called a *Fully Homomorphic Encryption* (FHE) scheme.

In 2009, Gentry [50] proposed the first fully homomorphic encryption scheme, signifying a new era in the research field of homomorphic encryption. Such a breakthrough invention allows all kinds of computations to be performed on encrypted data, without requiring decryption or any intermediate communication with the parties contributing the data. While partially homomorphic encryption schemes need to be combined with secure multiparty computation protocols to allow complete functionality, a fully homomorphic encryption scheme supports all operations to be performed locally, with no interaction. Although theoretically sound, the scheme of Gentry [50] is impractical. All this great functionality comes at the cost of a steeply increasing computation time, and an extensive enlargement of the size of the corresponding encrypted data. The first implementation of the FHE scheme of Gentry [50] was presented by Gentry, and Halevi [51]. This implementation is based on a modified version of the original encryption scheme, combined with other techniques introduced by Smart, and Vercauteren [92], and it remains highly inefficient for practical applications.

Between the two extremes of PHE and FHE, there are the so-called *Somewhat Homomorphic Encryption* (SHE) schemes. These schemes allow both multiplications, and additions to be performed on encrypted integers, but they only support a limited number of these operations on the ciphertexts. A notable SHE scheme that appeared quite early in time, is the scheme of Boneh et al. [12], which allows unlimited homomorphic additions to be performed on the ciphertexts, and a single multiplication. After the multiplication we can keep performing addition operations, albeit in a different group, which does not allow further multiplications to be performed.

Since the breakthrough of Gentry [50], many SHE, and FHE schemes followed. The most notable of these works include the schemes proposed by van Dijk et al. [96], Brakerski, and Vaikuntanathan [16, 17], Fan and Vercauteren [47], Brakerski [14], Brakerski et al. [15], Chillotti et al. [26], and Cheon et al. [25].

2.2 Multiparty Computation based on Secret Sharing

Secure Multiparty Computation (MPC) was first identified as an interesting problem to be researched by Yao [98]. We can describe MPC as the problem, where several participants wish to compute a function, and reveal only the result of this computation, on private variables contributed by all participants. The correctness of the computed (common) result must be ascertained, and the variables contributed as inputs by each participant should remain private (i.e.,

not be revealed to the other participants). A lot of fundamental work on secure multiparty computation, such as the works of Goldreich et al. [55], Chaum et al. [23], Beaver et al. [5], Beaver [3], Goldreich [53], and Cramer et al. [30], followed Yao's [99] seminal work. The field of Secure Multiparty Computation, especially in combination with Secret Sharing, and HE is still being actively researched, as it presents itself as an ideal solution to many interesting, real-life application scenarios, such as the ones discussed in Chapter 1. This is because MPC is significantly more efficient than HE solutions, which renders it practical. Furthermore, when together with the privacy of the underlying plaintext data, the correctness of the computed result is important, MPC can offer stronger guarantees than HE approaches, in that certain MPC protocols offer security against adversaries, who aim to manipulate the output of the computation.

Another reason to opt for MPC instead of HE approaches is that in several applications, the data to be contibuted to the secure computation does not come from a single individual, but from many individuals. In such applications privacy of the contributed data needs to be guaranteed, but the collective secure computation also needs to be deployed on a pool of data contributed by *several* individuals. MPC is therefore an approach that fits naturally this kind of applications. To overcome the limitations of traditional FHE in this context (i.e., the fact that the contributed plaintext data has to be encrypted under a single key, when standard FHE solutions are considered), we can deploy threshold FHE, where the private key is shared among the protocol participants, and hence the data does not need to be stored at a single place. The notion of multi-key FHE [73], is also a viable alternative to address the distributed computation scenario. However, the first attempts to construct multi-key FHE have been shown to be insecure [1]. In addition, most multi-key FHE schemes suffer from the inherent limitation of requiring all keys that will be involved in the secure computation to be known in advance, as the output cannot be combined with ciphertexts encrypted under other, new keys. Note, however, that this latter limitation, of knowing the protocol participants in advance, is also inherent in most MPC protocols.

When deploying a MPC solution, several considerations on the environment and requirements must be taken into account. The most prevalent such considerations are the following:

- The type of adversarial behavior that we need to combat. Are the adveraries only interested in learning private information about the users' private data, and intermediate computations, or are they willing to actively sabotage the computation by deviating from the protocol specification?
- The number of parties involved in the MPC protocol. How many parties participate in the MPC computation?

- The type of parties involved in the MPC protocol. Are all the parties participating actively in the computation, or do they simply contribute their data in the protocol (playing only the role of a data *dealer*)?
- The computational power of the MPC parties. Are all MPC parties equally powerful or do we need to distribute the computation in an unequal manner (e.g., if some of the parties are mobile nodes with limited capabilities)?
- Whether the MPC parties can begin their computation and communication, prior to the actual protocol execution, to create auxilliary material that can be used to speed up the so-called *online phase*, where the protocol specification and the inputs are known. This is called the *preprocessing model*, which once deployed can increase both the online, and the overal MPC efficiency (i.e., including the preprocessing phase).
- The communication efficiency. Do we need to account for as few communication rounds as possible (e.g., in network environments where synchronous communication is a challenge), or do we focus on decreasing the total communication cost (when the total bandwith is our main consideration)?

As efficiency is one of the main considerations for the deployment of MPC in real-life applications, next to provable security guarantees, most modern works (e.g., the works of Bendlin et al. [9], Damgård et al. [35, 33], Keller et al. [66, 67]) are based on the preprocessing model.

2.3 Multiparty Computation based on Garbled Circuits

Garbled Circuits (GC) is a special-case MPC approach. While HE solutions are non-interactive, in the sense that they do not require the data contributor to be online, or otherwise participate in the secure computation (yet they do incur a non-negligible communication cost), MPC solutions are interactive, requiring several communication rounds (in contrast to only two rounds for HE solutions), where the computing parties actively participate in the computation. GC is a meet-in-the-middle solution, where the GC protocols do require online, synchronous communication, but the protocol execution is completed in a *constant* number of rounds, and not in a number of rounds proportional to the depth of the circuit to be computed, as with standard MPC methods.

In fact, at the beginning of all MPC research, sits Yao's work [99], which is essentially a two-party garbling protocol. In the original two-party setting,

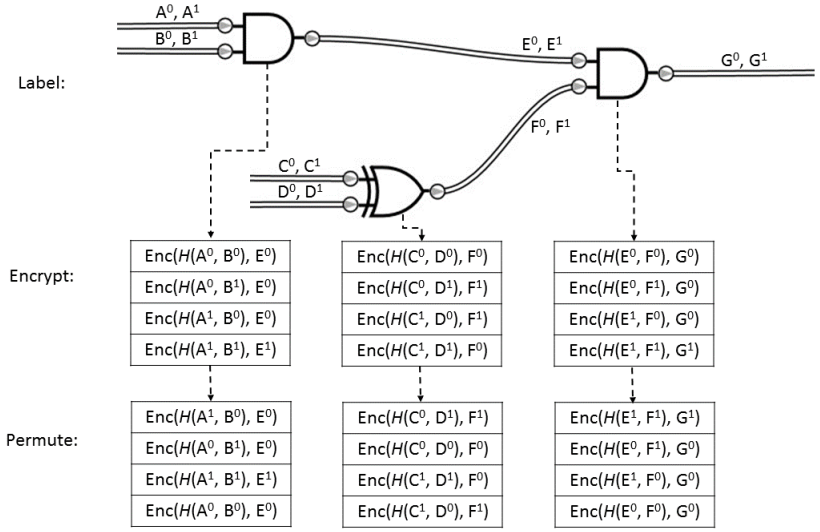


Figure 2.1: Garbling of a circuit with three gates.

a garbled circuit serves two parties, the *garbler*, and the *evaluator*, which work through the protocol in an asymmetric fashion. The garbler generates the "garbled" (i.e., randomized) version of the circuit, and hardwires their own inputs in the circuit. This garbling procedure is based on symmetric key encryption primitives, which makes it particularly efficient from a computational point of view. For all gates in the circuit, the garbler selects random labels for each entry, in each of the corresponding truth tables, hashes all possible combinations of these labels, and uses the hash as the symmetric key to encrypt the output value on this row of the truth table; then it permutes each truth table. The garbling process of a simple circuit is depicted in Figure 2.1. On receiving the garbled circuit, the (random) encoding of the garbler's inputs, and some encoding of the evaluator's input, the evaluator can walk through the circuit on a gate-by-gate basis to compute the final garbled result. At the end, the two parties communicate to reveal the plaintext output to both. More concrete techniques involved in the design of GCs are extensively discussed in Section 3.3.

The seminal work of Yao [99] was followed by the well-known BMR protocol [5]. In that work, Beaver et al. [5] show how to extend Yao's protocol from the two-party to the multiparty setting, and they introduce the *point-and-permute* technique, which allows the evaluator to decrypt only one out of the four received

ciphertexts in each truth table, while originally they would have to decrypt all four (and accept only the one that generates a valid message). To apply the point-and-permute technique, the garbler, in addition to the input and output labels, generates two *select bits* (aka *color bits*), encoding the real input value in a uniformly random fashion. The evaluator can now use these two color bits on each of the two inputs (per gate) to determine which row of the garbled table to decrypt. In a nutshell, with point-and-permute the garbler still has to send the whole truth table of each gate to the evaluator, but the evaluator can identify and decrypt only one of the four possible outputs in the truth table.

Naor et al. [79] were the first to suggest a so-called *row reduction* technique, which allows the GC to function according to the protocol specification, without sending all four rows of the truth table for each gate in the GC, but instead send only three of them. This is achieved by selecting one label in a way that forces the corresponding ciphertext to always be zero. The next breakthrough in the history of GC's is the *free-XOR* technique of Kolesnikov and Schneider [71]. As the name implies, with the free-XOR technique, there is no need to garble XOR gates in the circuit: the garbler does not send any garbled rows to the evaluator, and the evaluator does not therefore need to perform any decryption operations. This is achieved by selecting a uniformly random global difference for the output wire, select the zero output label of the gate at random, and then set the one output label to be the XOR of the global difference and the zero output label, i.e., $W^1 = W^0 \oplus \Delta$, for W^0, W^1 the zero and one output labels respectively, and Δ the global difference. Another row reduction technique followed in the timeline of GC's, by Pinkas et al. [83]. While this technique allows the elimination of two encrypted rows per truth table, instead of one (that the work of Naor et al. [79] could eliminate), it is not compatible with the free-XOR trick, which means that also XOR garbled gates cost two encrypted rows each (instead of zero with free-XOR).

Kolesnikov et al. [70] propose the *FleXOR* technique, which tries to get the most out of both the free-XOR, and the row reduction techniques. FleXOR manages to get to the optimal two encrypted rows per AND gate (like the work of Pinkas et al. [83]), while this flexible garbling allows also to get free XOR gates, but not always. The number of encrypted rows required for XOR gates in this setting, depends on the whole GC structure, and varies between zero and two. Focusing again on the optimization of AND gates, while maintaining compatibility with the free-XOR technique, Zahur et al. [100] designed the *half-gates* technique. The half-gates indeed achieved the optimal two encrypted rows per garbled AND gate, with compatibility with the free-XOR trick (i.e., zero encrypted rows for garbled XOR gates). We further elaborate on the half-gates construction in Section 3.3. In Table 2.1 we summarize the efficiency optimizations that garbling has undergone.

Technique	Size per gate		Encryption Calls per Gate			
			Garbler		Evaluator	
	XOR	AND	XOR	AND	XOR	AND
Classical [99]	4	4	4	4	4	4
Point-and-Permute [5]	4	4	4	4	1	1
Row Reduction 3 [79]	3	3	4	4	1	1
Free-XOR [71]	0	4	0	4	0	1
Free-XOR & Row Reduction 3	0	3	0	4	0	1
Row Reduction 2 [83]	2	2	4	4	1	1
FleXOR [70]	{0, 1, 2}	2	{0, 2, 4}	4	{0, 1, 2}	1
Half-Gates [100]	0	2	0	4	0	2

Table 2.1: Efficiency optimizations of Boolean Garbling. For each technique listed in this table, after the classical Yao's approach, we assume that the point-and-permute optimization is applied.

So far we have considered only Boolean GC's. Nowadays, however, arithmetic GC's are on the rise. Boolean circuits (garbled or not) are well suited to non-linear operations (e.g., comparison operations). Although it is of course possible to represent arithmetic inputs in a Boolean circuit, and apply linear functions on them, based on their binary representation, such an endeavor incurs a high computational and communication cost, compared to processing linear functions on arithmetic circuits. Most modern applications require a combination of linear and non-linear operations to be performed on the input data. The reason why arithmetic circuits prevail is that arithmetic circuits are significantly more efficient than Boolean circuits for the linear part, which usually presents itself as the bulk of the computation.

The first work to consider garbling of arithmetic circuits is the work of Ball et al. [2]. Firstly, Ball et al. [2] show how to extend the free-XOR technique of Kolesnikov and Schneider [71] from the Boolean to the arithmetic setting. This results in arithmetic GC's, where linear operations over the integers are for free. In addition, they leverage the Chinese Remainder Theorem (CRT) to represent the inputs and intermediate values in the circuit, achieving great perfomance gains over the straightforward conversion of integers to binary. Lastly, further optimizations are suggested by Ball et al. [2], based on a conversion method from CRT numbers to a positional number system, which allows non-linear operations to be performed more efficiently than the straightforward conversion to binary would allow.

While Ball et al. [2] show how to do two-party arithmetic garbling, Ben-Efraim [7] extends this to the multiparty setting. Ben-Efraim's approach [7] inherits the free addition property of its predecessor [2] (and extends this to more than two parties), and it also comprises a "designated" construction to garble multiplication gates, which is inspired by the half-gates technique [100]. Moreover, Ben-Efraim [7] suggests a construction for a garbled selector gate (i.e., a gate that given two arithmetic inputs and one binary input, outputs one of the arithmetic inputs, based on the value of the selection bit input), which reduces the computational cost at the evaluation stage by $\sim 33\%$, i.e., 2 decryption operations instead of 3, compared to the naive implementation of a selector gate.

Chapter 3

Cryptographic Primitives

In this chapter we present the main cryptographic primitives that are used throughout this thesis. We begin by an elaborate discussion on Secret Sharing, because our focus in on Secret-Sharing-Based MPC. We proceed with explaining Oblivious Transfer, and then we get deeper into the details of garbling, which has already been briefly introduced in Chapter 2. Finally, we present the three main primitives that allow us to construct MPC protocols with active security, namely Commitment Schemes, Message Authentication Codes, and Zero Knowledge Proofs. The discussion on the latter primitives is not formal, and only serves to explain the corresponding concepts.

3.1 Secret Sharing

Secret sharing refers to a cryptographic primitive, which allows a secret to be distributed among a group of participants. Once the secret has been distributed, each of the participants holds a *share* of the secret. When considering *secure* secret sharing, each share on its own should not reveal any information about the original secret. The secret can be reconstructed only if a sufficient number of individual shares are combined together. A bit more formally: a secret sharing scheme allows a *dealer* to distribute shares of a secret to n parties $P_1, \ldots, P_n$, such that only *authorized* subsets of these parties can reconstruct the secret, while any *unauthorized* subset of parties does not obtain any information about the secret. Secret sharing was invented independently, but in the same year (1979) by two different researchers: Adi Shamir [90], and George Blackley [10].

A *threshold* secret sharing scheme is a scheme where the authorized sets are all the sets whose size is at least some threshold value t. If there are n parties participating in the secret sharing and any t (or more) of them form an *authorized set* then we say that the scheme realizes the t-out-of-n *access structure*, where $1 \leq t \leq n$ is an integer; and call it (t, n)-threshold secret sharing scheme. When all parties' shares are required for the secret reconstruction, and no $t \leq n - 1$ parties are able to obtain any useful information about the secret, we refer to the secret sharing scheme as a scheme for the (n, n)-threshold access structure.

Easy examples of secret sharing methods include the *trivial secret sharing*, where the threshold $t = 1$, and the secret is handed in its entirety to all n parties. Another straightforward example is that of splitting a secret phrase or word in blocks of letters of predetermined size, and then distributing each block to one party. Consider for instance that one wishes to secret share the word "cryptography" to four parties, and they split it in the following way: "cry - - - - - - - - -", "- - - pto - - - - - -", "- - - - - - gra - - -", "- - - - - - - - - phy". In this case, an outsider possessing 0 shares has no information about the secret, other than that it consists of 12 letters. Outsiders would therefore need to guess the secret from the $26^{12} \simeq 95.5$ quadrillion possible combinations. On the other hand, a party with one share has a lot of information about the secret, and only needs to guess 9 out of the 12 letters, which is $26^9 \simeq 5.4$ trillion combinations. The more parties collude (and therefore combine their knowledge), the more the secret gets exposed. Clearly, these two examples of simple word concatenation, and trivial secret sharing are not secure; we use these examples here only for illustration purposes. Strictly speaking, a secure secret sharing scheme distributes shares so that anyone with fewer than t shares has no more information about the secret than someone with 0 shares.

In the cryptographic sense, we consider only *secure* secret sharing schemes. However, the level or type of security provided by each such secret sharing scheme differs. There are secret sharing schemes, such as Shamir's secret sharing [90], providing *information theoretic security* [91], and thus satisfying the property of *perfect secrecy*, and secret sharing schemes providing *computational security*. The various levels of security provided by cryptographic primitives, (e.g., the assumptions made on the power of the protocol adversaries) are further discussed in Section 4.

3.1.1 Linear Secret Sharing

Linear Secret Sharing Schemes (LSSS) are the schemes for which the reconstruction of the secret from the shares is a linear mapping. In practice, this means that any linear operations performed on the individual shares translates

to operations performed on the secret, upon reconstruction. This is a form of additive homomorphism, which allows meaningful (linear) operations to be performed indirectly on the secret, while actually being performed locally on the shares by each individual party, without requiring the exchange of additional information or other form of communication among the parties. It is this very property that makes LSSS a popular choice for constructing MPC protocols on top of them.

As the first to have been invented, but also satisfying strong security properties (namely perfect secrecy) along with the property of being linear, Shamir's secret sharing scheme [90] is perhaps the most popular secret sharing scheme to date. The goal of Shamir's secret sharing is to divide a secret S into n shares $S_1, \ldots, S_n$, such that if you know t or more S_i shares, S is easy to reconstruct, while if you know $t-1$ or fewer shares you gain no information about S over someone who possesses 0 shares. The intuition behind the scheme is that 2 points suffice to define a line, 3 points suffice to define a quadratic curve, 4 points suffice to define a cubic curve and so on. More generally, one needs t points to define a $t-1$ degree polynomial. For the scheme to be secure, the secret and shares are defined over a finite field $\mathbb{F}$ of size p, for p a prime number. To share a secret S, one chooses at random $t-1$ field elements $a_1, \ldots, a_{t-1}$, and let $a_0 = S$. They construct the polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{t-1} x^{t-1}$. Then, they find any n distinct points on the curve (except for the point $(0, S)$, for instance by setting $i = 1, \ldots, n$, and distributing to each party i the point $(i, f(i))$ (along with the public parameter p which defines the finite field). To reconstruct the secret we need any t of the points constructed in the previous step, and by using polynomial interpolation we can define the coefficients of the polynomial, including the constant term, which is the secret.

Another example of a LSSS, which is widely used in secure computation nowadays is *additive secret sharing*. Additive secret sharing, unlike Shamir's secret sharing is a scheme for the (n, n)-threshold access structure. Similarly to Shamir's secret sharing scheme, for traditional additive secret sharing the secret and shares are again defined over a finite field $\mathbb{F}$ of size p, for p a prime number. To split a secret S into n shares one chooses at random $n-1$ field elements $S_1, \ldots, S_{n-1}$, and lets $S_n = S - \sum_{i=1}^{n-1} S_i$. Each party then gets one of these shares. To reconstruct the secret all shares are required, and we simply need to add them to retrieve S.

3.1.2 Verifiable Secret Sharing

Verifiable Secret Sharing [27] yields another class of secret sharing schemes, which allow proving some property of a (secret shared) message, without revealing

the message itself. A milestone work in the field of verifiable secret sharing is the work of Gennaro et al. [49]. Applications of (publicly) verifiable secret sharing schemes made possible the construction of the first secure electronic voting mechanisms, and platforms [88]. Other practical applications of this special type of secret sharing include collaborative supply chain management, where competing parties should collaborate by combining their private sensitive data, so as to optimize the supply and demand configuration [37].

3.1.3 Applications of Secret Sharing

Beimel lists numerous cryptographic applications of secret sharing in their survey [6], including secure information storage, Byzantine agreement, threshold cryptography, access control, attribute-based encryption, and generalized oblivious transfer. However, the most prominent such application is MPC. Building MPC protocols on top of secret sharing primitives allows us to devise practical (in terms of efficiency) privacy-preserving protocols for a variety of real-life problems: from private genomics, to private machine learning applications. There is a plethora of MPC protocols to choose from, based on the scenario at hand, and the resources available.

3.2 Oblivious Transfer

Oblivious Transfer (OT) is a protocol executed between two parties: a *sender* and a *receiver*. The most straightforward OT protocol is the so-called 1-out-of-2 OT, which allows the sender to transfer *one* piece of information to the receiver, out of *two* possible values. More generally, OT allows the sender to transfer *some* out of *many* pieces of information to the receiver. During this protocol, the sender does not learn anything about which pieces of information were actually transferred to the receiver; the receiver, on the other hand, does not gain any information about the pieces they did not receive. The notion of OT dates back to the work of Rabin [84], and it is a protocol that requires asymmetric cryptographic primitives to be deployed for its implementation. In this seminal work [84], which is based on the RSA cryptosystem [87], the sender sends only one message to the receiver, with probability 1/2, and remains oblivious as to whether the message was actually received.

Figure 3.1: 1-out-of-2 Oblivious Transfer.

3.2.1 1-out-of-2 OT

The first *1-out-of-2 OT* scheme, i.e., a protocol where the sender transfers two values, and the receiver only obtains one of them, is proposed by Even et al. [46]. This primitive, illustrated in Figure 3.1, gives rise to the construction of MPC protocols. More importantly, it has been shown that OT is complete for MPC [55, 68], allowing to evaluate any function, given only a secure OT implementation, and no additional cryptographic primitives. Recall, however, that OT's are based on asymmetric primitives for their implementation, which renders them inefficient, since the oblivious transfer of a single bit requires two encyption, and one decryption operations, as well as the transfer of the two large ciphertexts over the network.

3.2.2 1-out-of-n OT

A *1-out-of-n OT* protocol is a generalization of the standard 1-out-of-2 OT, where the sender has n values (instead of 2 in the previous construction), and the receiver has a selection index $i, 0 \leq i < n$, allowing them to obtain the i^{th} value. From this protocol execution the sender learns nothing about the selection index i of the receiver, and the receiver learns the i^{th} value and nothing about the $n-1$ other values of the sender that they did not select. It is possible to construct a 1-out-of-n OT, based on a 1-out-of-2 OT, in a straightforward manner, by executing n parallel 1-out-of-2 OT's. However, this would incur computational and communication cost linear in n. Naor and Pinkas [78] constructed a 1-out-of-n OT with complexity logarithmic in n.

3.2.3 k-out-of-n OT

Further generalizing the standard 1-out-of-2 OT primitive, Ishai and Kushilevitz [61] propose an oblivious polynomial evaluation construction, which gives rise to new OT primitives, namely *generalized-OT* (GOT). The proposed GOT

can be regarded as a *k-out-of-n OT*, where the receiver obtains k messages out of the n messages of the sender. The security requirement here is an extension of the previous one, where the sender learns nothing at all from the protocol, and the receiver learns the k messages, and nothing about the other $n - k$ messages of the sender.

3.2.4 OT Extension

A lot of research has been focused on improving the OT primitive, because on the one hand it is a complete primitive for MPC, and on the other hand it is an expensive primitive, since it is based on asymmetric cryptographic primitives. Many works have considered *extending* OT to make it practically efficient in the amortized sense, where the protocols incorporating the primitive require (a lot) more than a single bit to be transferred obliviously. The first OT extension protocol is due to Beaver [4], while the most notable one is that of Ishai et al. [60]. In this latter work, Ishai et al. [60] show how to perform only a few OT's from scratch (i.e., using the standard expensive methodologies), and then be able to perform many additional OT's at the cost of a constant number of invocations of relatively inexpensive symmetric key primitives. While the first protocol of Ishai et al. [60] is passively secure, they also propose an actively secure version of their protocol, which comes with an increased cost by a factor σ, for σ a statistical security parameter. More recently, Keller et al. [65] presented an actively secure OT extension protocol, where malicious security comes at negligible extra cost (compared to the cost of the passively secure protocol).

3.2.5 Oblivious Polynomial Evaluation - OPE

Getting an insight on why OT is complete for MPC, let us explain how we can use OT to compute the AND operation between two bits held by two different parties, where the result of the operation is additively secret shared. The sender holds a bit s, and the receiver holds a bit r; together they wish to compute the binary product $p = s \wedge r = s' \oplus r'$ in a secret shared form, where each of them holds a share of the XOR operation over p at the end of the protocol. To compute this, the sender selects a random masking bit s_0, and masks their input with it by setting $s_1 = s \oplus s_0$. Then the sender inputs to the OT s_0, and s_1, while the receiver inputs r. The output of the OT will then be s_r for the receiver. The receiver then sets the output of the secure AND operation to be $r' = s_r$, and the sender sets their output to be $s' = s_0$. The protocol above is

correct because $r' = s_r = r \wedge s_1 \oplus (1 \oplus r) \wedge s_0 = r \wedge (s_1 \oplus s_0) \oplus s_0$, and if we XOR the two outputs, we indeed get: $s' \oplus r' = s_0 \oplus s_r = r \wedge s$.

The aforementioned simple example gives rise to the so-called *Oblivious Polynomial Evaluation* (OPE) primitive, which has been studied already by Naor and Pinkas in 1999 [78]. The well-known Gilboa's multiplication protocol [52] is based on the same concept. In modern MPC applications a closely related primitive has been suggested, namely the primitive of *Correlated Oblivious Product Evaluation* (COPE) [66]. The COPE protocol presented by Keller et al. [66] is essentially a generalization of Ishai et al.'s protocol [60] from the binary to the arithmetic case, and it is also inspired by Gilboa's multiplication [52]. Similarly to the previous example, there are two parties who wish to obtain an additive sharing of the product $x \cdot \Delta$, where the sender holds x, and the receiver holds Δ. To accommodate the arithmetic (instead of binary) inputs, the parties run k sets of 1-out-of-2 OT's for k-bit inputs. The proposed product evaluation is correlated in the sense that the one party's input Δ is fixed at the beginning of the protocol for many protocol runs. After a one-time expensive initialization of the COPE protocol, the extension step, generating fresh OT's without the extensive costs incurred by the asymmetric key primitives, can be repeated several times on new inputs.

3.3 Garbling

Having discussed in Section 3.2 OT as a primitive, allows us to elaborate further on the concrete techniques used for garbling. In Section 2.3 we have introduced garbled circuits as a secure computation methodology and we have briefly explained the most notable optimizations that GC has undergone throughout history. In this section we give more details on how garbling is concretely deployed, as well as on the half-gates technique, and multiparty garbling, which are relevant to the optimizations presented in Chapter 10.

Recall that a GC is a randomized version of a circuit allowing traditionally two parties to evaluate a function on the union of their private inputs, without revealing anything more about their private inputs than what can be inferred from their own inputs and the output alone. While in the two-party setting the garbling procedure is asymmetric, between the two parties called the garbler and the evaluator, in the multiparty setting all parties play the role of the garbler, and any parties can play the role of the evaluator. We now focus on gates receiving two inputs and having one output wire. Each such fan-in-2 gate $g : \mathbb{F}^2 \to \mathbb{F}$ in the circuit has two input wires, denoted u and v, and one output wire denoted w. The garbled gate is expressed as a truth table with one row

for each input combination $(\alpha, \beta) \in \mathbb{F}^2$, and the corresponding output. Each row has thus the following form: $(\alpha, \beta, g(\alpha, \beta))$, as shown also in Table 3.1a for the case of arithmetic garbling.

The garbler samples a random key for each possible value of α, β and $\gamma = g(\alpha, \beta)$. These keys typically live in a finite field extension of the basic field $\mathbb{F}^\ell$, where ℓ serves as a security parameter mandating the length of the keys. The length of the keys depends on the GC implementation, and it is not predetermined. However, some garbling optimizations do impose particular requirements on the encryption scheme used. The real input values α, β in the truth table are replaced in the garbled version by the corresponding encryption keys, as it is shown in the first and second column of Table 3.1b. The keys corresponding to the output wire w are double encrypted under both of the corresponding input keys, using a *Pseudorandom Function* (PRF) on input the gate index g. Finally, the key of the output wire w is added to create the ciphertext:

$$\widetilde{g}_{\alpha,\beta} = F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + \mathsf{k}_{w,g(\alpha,\beta)}.$$

These ciphertexts form the last column of the garbled truth table (see Table 3.1b), and they are sent to the evaluator, once they are permuted.

The garbler's inputs are hardwired in the GC. For instance, if the input of the garbler to a gate is the u wire, then they perform the encryption only under the keys corresponding to the v inputs. What still remains to be hidden is the order of the ciphertexts in the table, as this would reveal the garbler's inputs to the evaluator. Thus, to protect the garbler's input from the evaluator the ciphertexts are randomly permuted using so-called *permutation* or *masking* values chosen by the garbler, and denoted by λ. In the binary circuit case, the permutation values are random bits, while in the arithmetic circuit case we aim at acquiring a rotation of the table rows. The permuted, based on the masking values, garbled table that is sent to the evaluator is shown in Table 3.1c.

For the GC evaluation phase, the evaluator receives the keys corresponding to its inputs, via OT, so that the garbler remains oblivious to the evaluator's inputs, while the evaluator does not learn the keys for any values other than the ones corresponding to its own inputs. Using these keys, the evaluator decrypts gates as follows: $\widetilde{g}_{\alpha,\beta} - F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g)$. The result of this decryption is a key that can be used to decrypt the next gate in the circuit, together with the corresponding output key of the next gate, which the evaluator has received from the garbler. In this gate-by-gate manner the evaluator walks through the GC, until the key of the final output gate is obtained.

To correctly evaluate the garbled gates, the evaluator needs next to the output key, also a so-called *external* or *signal* value, denoted by e, which is the real value v, masked with the masking value λ; that is, $e = v + \lambda$. This is because

u	v	$w = g(u, v)$
0	0	$g(0,0)$
...	...	...
0	$p-1$	$g(0,p-1)$
...	...	...
$p-1$	0	$g(p-1,0)$
...	...	...
$p-1$	$p-1$	$g(p-1,p-1)$

(a) Truth Table of an arithmetic fan-in-2 gate with input and output wires $\in \mathbb{F}_p$.

$\mathsf{k}_{u,\alpha}$	$\mathsf{k}_{v,\beta}$	$\widetilde{g}_{\alpha,\beta} = F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + \mathsf{k}_{w,g(\alpha,\beta)}$
$\mathsf{k}_{u,0}$	$\mathsf{k}_{v,0}$	$\widetilde{g}_{0,0} = F_{\mathsf{k}_{u,0},\mathsf{k}_{v,0}}(g) + \mathsf{k}_{w,g(0,0)}$
...	...	...
$\mathsf{k}_{u,0}$	$\mathsf{k}_{v,p-1}$	$\widetilde{g}_{0,p-1} = F_{\mathsf{k}_{u,0},\mathsf{k}_{v,p-1}}(g) + \mathsf{k}_{w,g(0,p-1)}$
...	...	...
$\mathsf{k}_{u,p-1}$	$\mathsf{k}_{v,0}$	$\widetilde{g}_{p-1,0} = F_{\mathsf{k}_{u,p-1},\mathsf{k}_{v,0}}(g) + \mathsf{k}_{w,g(p-1,0)}$
...	...	...
$\mathsf{k}_{u,p-1}$	$\mathsf{k}_{v,p-1}$	$\widetilde{g}_{p-1,p-1} = F_{\mathsf{k}_{u,p-1},\mathsf{k}_{v,p-1}}(g) + \mathsf{k}_{w,g(p-1,p-1)}$

(b) All possible combinations of the secret keys on the input wires, for real input values α, and β, together with the corresponding garbled output presented in the third column.

$\widetilde{g}_{\alpha,\beta} = F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + (\mathsf{k}_{w,g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w} \| g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w)$
$\widetilde{g}_{0,0} = F_{\mathsf{k}_{u,0},\mathsf{k}_{v,0}}(g) + (\mathsf{k}_{w,g(0-\lambda_u,0-\lambda_v)+\lambda_w} \| g(0-\lambda_u,0-\lambda_v)+\lambda_w)$
...
$\widetilde{g}_{0,p-1} = F_{\mathsf{k}_{u,0},\mathsf{k}_{v,p-1}}(g) + (\mathsf{k}_{w,g(0-\lambda_u,p-1-\lambda_v)+\lambda_w} \| g(0-\lambda_u,p-1-\lambda_v)+\lambda_w)$
...
$\widetilde{g}_{p-1,0} = F_{\mathsf{k}_{u,p-1},\mathsf{k}_{v,0}}(g) + (\mathsf{k}_{w,g(p-1-\lambda_u,0-\lambda_v)+\lambda_w} \| g(p-1-\lambda_u,0-\lambda_v)+\lambda_w)$
...
$\widetilde{g}_{p-1,p-1} = F_{\mathsf{k}_{u,p-1},\mathsf{k}_{v,p-1}}(g) + (\mathsf{k}_{w,g(p-1-\lambda_u,p-1-\lambda_v)+\lambda_w} \| g(p-1-\lambda_u,p-1-\lambda_v)+\lambda_w)$

(c) Permutation of the garbled output presented in Table 3.1b, so that it can be sent to the evaluator without revealing information about the order of the ciphertexts.

Table 3.1: Arithmetic garbling representation via the Truth Tables' transition.

the evaluator needs to learn which ciphertexts to decrypt for each gate, despite the rows of the garbled table being permuted. The ciphertexts are then

$$\widetilde{g}_{\alpha,\beta} = F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + \left(\mathsf{k}_{w,g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w} \middle\| (g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w\right),$$

where $g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w$ is the external value representing the masked output wire, as shown in Table 3.1c. This is a generalized representation of

Beaver et al.'s point-and-permute technique [5], which applies both to binary and to arithmetic GC's.

3.3.1 Half-Gates

The half-gates technique, introduced by Zahur et al. [100], is the first garbling technique that allows AND gates to be garbled at the cost of two ciphertexts (instead of the straightforward implementation incurring a communication cost of four ciphertexts), while being compatible with the free-XOR technique. The half-gates technique extends from the binary to the arithmetic garbling case, where the garbling of a fan-in-2 multiplication gate with inputs in $\mathbb{F}_p$ requires $2p$ ciphertexts, instead of p^2 ciphertexts that straightforward garbling would require. The idea behind the half-gates is to exploit the asymmettry between the information held by the garbler and the evaluator, in order to garble two (asymmetric) gates, the addition of which is the desired multiplication result.

Informally, for a multiplication gate with input wires u and v, and output wire w, where we denote again the signal value as e, the real value as v, and the permutation element as λ, we aim to have computed at evaluation time the external value:

$$e_w = v_w + \lambda_w = v_u v_v + \lambda_w,$$

and the corresponding keys. To this end, the first half-gate aims at computing $v_u(v_v + \lambda_v)$, and the second half-gate computes $-\lambda_v v_u$.

More formally, to design a half-gate, we observe what can be obtained from products of external values with keys of input wires, i.e., from $e_u \cdot \mathsf{k}_{v,e_v}$, or $e_v \cdot \mathsf{k}_{u,e_u}$. This is useful, because at the evaluation phase, the evaluator can compute the product of an external value e_u with a key k_{v,e_v} to obtain a value related to the product $v_u \cdot v_v$, and then it can correct the errors incurred from the masking values using garbled gates. Concretely, such a product gives us:

$$\begin{aligned} e_u \mathsf{k}_{v,e_v} &= (v_u + \lambda_u)(\mathsf{k}_{v,0} + (v_v + \lambda_v)) \\ &= v_u \mathsf{k}_{v,0} + \lambda_u \mathsf{k}_{v,0} + v_u v_v + \lambda_u v_v + v_u \lambda_v + \lambda_u \lambda_v \\ &= v_u v_v + \underbrace{v_u \mathsf{k}_{v,0} + v_u \lambda_v}_{\text{Dependent on } v_u} + \underbrace{\lambda_u v_v}_{\text{Dependent on } v_v} + \underbrace{\lambda_u \mathsf{k}_{v,0} + \lambda_u \lambda_v}_{\text{Dependent on neither}} \end{aligned}$$

Recall that our goal is to obtain the output key $\mathsf{k}_{w,e_w} = \mathsf{k}_{w,0} + (\lambda_w + v_u v_v)$. In this expression the challenging computation is the product $v_u \cdot v_v$. The half-gates technique exploits the fact that at evaluation time the product $e_u \mathsf{k}_{v,e_v}$ is easy to be computed by the evaluator, and at the same time it contains the challenging term $v_u \cdot v_v$ we are after. The remaining garbling aims at correcting the errors

introduced by the computation of $e_u \mathsf{k}_{v,e_v}$, which incurs a lower communication cost than straightforward garbling of multiplication gates, because these errors are functions in the value of only *one* of the two real wire values v_u or v_v together with the masking values, which are predetermined. This means that the ciphertexts containing the "corrections" can be generated independently for each pair of inputs in $\mathbb{F}_p^2$, requiring only $p + p$ ciphertexts, instead of $p \cdot p$ ciphertexts that would be required in the conventional garbling manner (i.e., for all possible cross-terms).

To obtain $\mathsf{k}_{w,e_w} = \mathsf{k}_{w,0} + (\lambda_w + v_u v_v)$, for every $\gamma \in \mathbb{F}_p$, the garbler generates two ciphertexts: one encrypting

$$\mathsf{k}_{w,\mathsf{g},0} + \lambda_w - ((\gamma - \lambda_u)(\mathsf{k}_{v,0} + \lambda_v) + (\lambda_u \mathsf{k}_{v,0} + \lambda_u \lambda_v)) \,,$$

and the other encrypting

$$\mathsf{k}_{w,\mathsf{e},0} - (\gamma - \lambda_v)(\lambda_u) \,.$$

The output wire key is set to $\mathsf{k}_{w,0} := \mathsf{k}_{w,\mathsf{g},0} + \mathsf{k}_{w,\mathsf{e},0}$. The evaluator decrypts the ciphertexts corresponding to $\gamma = e_u$ for the first half gate, and $\gamma = e_v$ for the second. Since $e_u - \lambda_u = v_u$ and $e_v - \lambda_v = v_v$, the correct key can be obtained by summing the two resulting plaintexts and the value $e_u \mathsf{k}_{v,e_v}$. In the original two-party garbling, one gate input comes from the garbler and the other from the evaluator, meaning that the evaluator is also involved in the garbling of the half gates. This results in reduced communication, because each party knows one of the wire masks. In the multiparty setting, described in the subsequent section, no party knows the wire masks, which makes the setting more challenging. Still, the half-gates technique in the multiparty setting, reduces the communication cost, in terms of number of ciphertexts that need to be sent from quadratic (p^2) to linear ($2 \cdot p$).

3.3.2 Multiparty Garbling

In multiparty garbling, which is due to Beaver et al. [5], all parties act as garbler and evaluator. Using MPC, each party generates keys for a circuit, and the masking values are randomly chosen and secret shared among the parties (i.e., no single party knows the masking values). This way, for each gate, each party holds n ciphertexts, indexed by j:

$$\widetilde{g}^j_{\alpha,\beta} = \sum_{i=1}^{n} F_{\mathsf{k}^i_{u,\alpha},\mathsf{k}^i_{v,\beta}}(g,j) + \left(\mathsf{k}^j_{w,0} + (g(\alpha - \lambda_u, \beta - \lambda_v) + \lambda_w)\right).$$

Each party P_i generates one set of keys, indexed by i. The external values on the wires can be learnt by each party, by looking at the output plaintext

m^i from its own circuit; then they set $e_w = (m^i - \mathsf{k}^i_{w,0})$ and $\mathsf{k}^i_{w,e_w} = m^i$. The aforementioned construction is a generalization of garbling protocols over $\mathbb{F}_2$. Notice, however, that for a Boolean circuit, the half-gate technique is no more efficient than the straightforward approach, except for the two-party setting, where one party is the garbler and the other the evaluator, rather than all being both, as in the multiparty setting.

3.4 Ingredients for Actively Secure MPC

This section presents the three main ingredients that allow us to construct MPC protocols, secure in the presence of an active adversary, namely *commitment schemes*, *Message Authentication Codes* (MACs), and *Zero Knowledge Proofs* (ZKP). A more elaborate discussion on the active security model itself, and the involved adversary types is given in Chapter 4.

3.4.1 Commitment Schemes

A commitment scheme is a cryptographic primitive, allowing a party to commit to a certain value, while keeping it secret from the other parties in the protocol. The commitment scheme serves to ensure that once committed, the parties can no longer alter their inputs, upon observing the inputs of the other protocol participants. The concept of commitments was introduced by Brassard et al. [18]. Cryptographic commitment schemes need to satisfy two properties:

1. *Binding:* intuitively this property ensures that a party cannot change a value or statement after they have committed to it; that is the party cannot produce a proof that they commited to a value or statement different from the original one. More formally, revealing two commitments $\mathsf{Commit}(m)$, $\mathsf{Commit}(m')$, where $m \neq m'$, should yield two different values.
2. *Hiding:* the commitment should not reveal any information about the corresponding value or statement. More formally, two commitments $\mathsf{Commit}(m)$, $\mathsf{Commit}(m')$ are indistinguishable to an adversary.

A commitment scheme is executed in two phases:

1. the *commit phase*, when the value or statement is chosen and specified;
2. the *reveal phase*, when the value is revealed and checked.

3.4.2 Message Authentication Code

A message authentication code (MAC) is a short piece of information, often referred to as the *MAC*, which serves –as the name implies– to authenticate a message. Authentication means that the MAC ensures that the message has not been altered, and that it indeed comes from the sender who claims to have sent it. MACs are a symmetric key primitive, and thus require the communicating parties to have established a common secret key prior to the actual communication, which makes them different from digital signatures. For a detailed discussion on MACs, we refer the reader to the book of Goldreich [54]. The use of MACs in the context of actively secure MPC is further discussed in Chapter 5.

3.4.3 Zero Knowledge Proofs

A zero-knowledge proof is a protocol between two parties, called the *prover* and the *verifier*. In such protocols, the prover aims to convince the verifier that they know a secret witness (e.g., secret key) for a public statement (e.g., public key) without disclosing any information about the witness. The essence of zero-knowledge lies in that at the end of running the protocol, the verifier learns nothing beyond the truth of the statement. ZKP systems were first introduced by Goldwasser et al. [57].

A ZKP system is expected to satisfy the following properties:

1. *Completeness:* The property of completeness ensures that if the statement of the prover is true, a verifier will be convinced by the protocol, and thereby will accept the proof. This is true under the assumption that prover and verifier are following the protocol honestly.

2. *Soundness:* The property of soundness guarantees that if the statement of the prover is false (i.e., the prover attempts to cheat), they cannot convince an honest verifier except with negligible probability. This protects the verifier from a malicious prover.

3. *Zero-Knowledge:* The property of Zero-Knowledge ensures that a verifier learns nothing beyond the fact that the statement is true. This is usually formalized by constructing a simulation algorithm that simulates the proof without knowing the witness.

A *Zero-Knowledge Proof-of-Knowledge* (ZKPoK) system is a special case of ZKP systems, where the verifier does not learn anything about the witness

associated with the prover's statement, other than the fact that the prover *knows* the witness.

Proof-of-Knowledge (PoK) protocols can be either interactive or non-interactive. In an interactive PoK system, which is also are known as a Σ-protocol (Sigma Protocol), the prover and verifier interact in several rounds. Such protocols need to satisfy Completeness, Special-Soundness, and Honest-Verifier-Zero-Knowledge (HVZK) that are defined as follows:

1. *Special-Soundness:* Special soundness guarantees that if the prover does not know the witness, they cannot convince the verifier about the public statement. In Σ-protocols this is formalized by showing that there exists an efficient extraction algorithm, so-called *extractor*, whuch given two acceptable tuples of the protocol, can extract the witness.

2. *Honest-Verifier-Zero-Knowledge:* The property of zero-knowledge ensures that if the prover has generated the proof honestly, the protocol transcript does not reveal any information about the witness to the verifier. In the case of Σ-protocols, this property is formalized by constructing a polynomial-time algorithm, so-called *simulator*, which given an honestly generated random challenge by the verifier, it can generate simulated proofs that are indistinguishable from the real ones. Due to the need for an honestly generated challenge by the verifier, this notion is called HVZK, and it is weaker than the standard ZK.

Chapter 4

Provable Security

This chapter details how we model the adversaries acting against our MPC protocols, based on their different behaviors, and the assumptions on their powers. Then, it explains how security proofs are constructed, and focuses on the Universal Composability framework, which is the most commonly used framework to prove MPC protocols secure, during the last decade.

4.1 Adversary Models

First, we classify potential adversaries based on their behaviors and goals: Do they wish only to learn information about the private inputs that the protocol handles, or are they willing to actively deviate from the protocol to sabotage its execution? Then, we discuss how many parties an adversary can corrupt, and when they can be corrupted. Before we proceed with proving security, it is also important to establish the assumptions on the computational resources available to the adversary.

4.1.1 Adversarial Behaviors

The most commonly assumed adversarial behaviors fall under one of the following two categories: passive adversaries, and active adversaries. However, there have been also other types of behaviors identified in the literature, lying somewhere in between these two extremes, with the most notable one being covert adversaries.

- *Passive* adversaries, also known as *semi-honest* or *honest-but-curious* adversaries are assumed to be able to corrupt parties in a way that allows them to read the version of the protocol transcript of these corrupted parties, but all protocol participants (even the corrupted ones) are assumed to follow the protocol instructions correctly. Using the transcripts of the corrupted parties, the passive adversary tries to learn as much information as possible about the private protocol inputs, hence the name *honest-but-curious*. Although this adversarial model allows us to build rigorous security proofs, the assumption of following the protocol instructions honestly is strong. To build practical protocols in terms of efficiency, this was the standard adversarial model used in MPC in the previous decade. Nowadays, this model is considered weak.

- *Active* adversaries, also known as *malicious* adversaries are the other extreme. This type of adversary can corrupt parties in a way that allows them to arbitratily deviate from the protocol instructions. Given the weaker assumption of active corruptions, this adversarial model provides stronger security guarantees. Clearly, the adversary is more realistic in this model, and this is the reason why MPC protocols nowadays aim to be proven maliciously secure.

- *Covert* adversaries are assumed to be able to act maliciously, but when they do so there is a certain probability that they will get caught. This is essentially an attempt to bridge the two exremes of passive and active adversaries, so that MPC protocols can be tailored to the security-efficiency tradeoff that each application requires. This setting aims at modeling a realistic adversary, where being caught comes with a penalty, therefore minimizing the incentive for the adversary to cheat the more the probability of getting caught increases.

4.1.2 Corruption Thresholds

One can assume any number of corruptions possible, as long as it applies to a strict subset of the protocol participants. We denote by n the total number of protocol participants, and by t the bound on the maximum number of corruptions allowed by the MPC protocol. Although we can have any number of corruptions, as long as $t \leq n - 1$, the most notable classes of corruption thresholds are the following, ordered from the least to the most secure:

- *Honest Supermajority* refers to the case where $t < n/3$, which means that the adversary can corrupt strictly fewer than one third of the protocol participants.

- *Honest Majority* refers to the corruption threshold $t < n/2$, where as the name implies, the adversary can corrupt strictly fewer than half of the parties.
- *Dishonest Majority* is the complementary of the previous case, where $t \geq n/2$.
- Finally, the most stringent setting is called *full-threshold*, and it refers to the case where the adversary is allowed to corrupt all but one of the protocol participants, i.e., $t = n - 1$.

The special case of two-party computation is worth mentioning in this context, as the full-threshold corruption, coincides with the dishonest majority corruption, and essentially refers to the bound of a single corrupted party.

4.1.3 Timing of the Corruptions

Based on *when* an adversary is allowed to fix their set of corrupted parties, we discern two types of adversaries:

- *Static adversaries* are assumed to decide on the set of corrupted parties, and fix it, prior to the protocol execution. The sets of honest and corrupted parties cannot be changed throughout the computation.
- *Adaptive adversaries* on the other hand are more powerful, having the freedom to arbitrarily decide the particular parties they wish to corrupt, and when to do so, throughout the whole protocol execution.

4.1.4 Computational Power of an Adversary

Adversaries are further classified based on the computational power we assume they have access to, to attack our MPC protocols. The two main classes of adversaries in this context are:

- *Computationally-bounded* adversaries, are assumed to be able to execute only polynomially many (randomized) steps, and as such they are also known as *probabilistic polynomial-time* (PPT) adversaries. This is an attempt to realistically model the possible adversaries, as in practice an adversary can only have access to limited computational power. In MPC protocols and their security proofs we consider a so-called computational security parameter, usually denoted by κ, to quantify the computational power of such an adversary.

- *Computationally-unbounded* adversaries, as the name implies, are assumed to have limitless computational resources. Although this adversary type is too strong to exist in practice, this assumption serves for proving protocols and cryptographic primitives secure. Concretely, protocols and primitives proven secure under this assumption are shown to be *unconditionally secure*, or *information theoretically secure*, and they enjoy the property of *perfect secrecy*.

4.2 Security Proofs

Commonly, proving security of a cryptosystem is based on a reduction of its security to a hardness assumption. This means that we show that a cryptosystem is secure, if the hardness assumption holds. Relying on well established hardness assumptions, instead of attempting to prove each individual primitive or cryptosystem secure, adds to the credibility of the proofs, as the hardness assumptions withstand years of cryptanalysis. In this section we review the two main approaches to proving security: game-based, and simulation-based proofs, focusing on the latter, which is the most common technique used to prove the security of MPC protocols.

4.2.1 Game-Based Security Proofs

Game-based security proofs are designed in the form of a game, where an *adversary* plays against a *challenger*. The goal of the adversary in this game is the condition we are trying to prove that does not hold (i.e., breaking the cryptosystem). The challenger provides the adversary with the view of the game, based on the information that the cryptosystem would allow external parties to observe. The proof in the end shows that the probability of an adversary winning the game is negligible with respect to a predefined security parameter.

4.2.2 Simulation-Based Security Proofs

In the simulation-based security paradigm, we distinguish between two worlds: the *real* and the *ideal* world. In the real world, the protocol is executed interactively among the participants, while the adversary can observe all public values exchanged, and all private values that corrupted parties input. In the active security model, the adversary can also force the corrupted parties in the real world to input whatever the adversary wishes. In the ideal world, the protocol participants interact only with a *trusted party* instead of exchanging

values with each other. Concretely, they query the trusted party, which is designed to provide them with the correct answers or results, based on their predefined functionality. To write a simulation-based proof, we need to construct a *simulator*, showing that the protocol execution in the real world is indistinguishable from the protocol execution in the ideal world.

This proof technique is successful, if we can show that the view of the adversary in the real world is indistinguishable from their view in the ideal world, where a trusted party is assumed. Then, the security of the protocol at hand is guaranteed under the given assumptions. This is because indistinguishability, in this context, shows that the adversary cannot derive any information from observing the corrupted parties' private values, or the public values of the protocol transcript. In practice, we aim at constructing a simulator in the ideal world, interacting only with the trusted party. For the messages that differ in the ideal and real world, to win the game, the adversary should be able to distinguish whether they are interacting with the simulator or with the real world. The reduction here aims at showing that distinguishing between these two views, would break one of the hardness assumptions we have made.

4.2.3 Modular Composition

The description of simulation-based proofs, given in Section 4.2.2 assumes that the cryptographic primitive, or MPC protocol to be proven secure is executed in isolation. Canetti [21] proved that the same security guarantees hold, even if the MPC protocol is executed as part of a larger computation. The so-called *modular composition* theorem allows us therefore to prove secure smaller subprotocols that can be invoked by other, possibly secure, protocols, while the security of the whole system is still guaranteed. However, this *sequential composition* of protocols still assumes that the protocols are executed in the stand-alone setting, meaning that no other messages are being sent, and no other secure protocols are being executed at the same time.

4.2.4 Universal Composability

Although modular sequential composition is a very strong tool to build secure systems composed by different subprotocols, excluding concurrent protocol execution is a critical limitation. In practice, we wish to be able to prove security even when protocols are executed concurrently and other messages are also exchanged between the protocol participants at the same time. In fact, it is desirable to parallelize as much of the computation as possible for efficiency reasons, without negatively affecting security. To this end, Canetti [22]

introduced the notion of *universal composability* (UC) and showed that protocols proven secure in this framework, remain secure even when executed in parallel with other (secure) protocols, or other instances of themselves.

To serve this more stringent UC framework, the simulation-based paradigm of the stand-alone setting is augmented with an additional entity called the *environment*. In the universal composability setting, the ideal vs. real world views should remain indistinguishable to the adversary, but they should also be indistinguishable to the environment. The environment is yet another observer, like the adversary, but its powers are enhanced. The environment is allowed to interact with the adversary, and force them to take action. In addition, the environment may force the inputs of the honest parties, and observe their outputs. However, the environment is assumed to not have access to any intermediate results or communications between the honest parties. Essentially, the environment tries to capture anything external to the protocol during a real execution, to serve the UC framework in that security is preserved, regardless of other corrupted entities that may be observing the parallel executions.

Chapter 5

MPC in the Preprocessing Model

In this chapter we discuss the construction of MPC protocols in the *preprocessing model*. The preprocessing model is the most common way to construct MPC protocols in the last decade. We focus on this paradigm, because the contributions of this thesis are based on the preprocessing model as well. The reason why this paradigm has become so popular is because it increases the efficiency of MPC protocols. In the preprocessing model the MPC computation is divided into two parts: the *offline* or *preprocessing* phase, and the *online* phase. The offline phase is input independent, and entails the evaluation of the desired function on random inputs, which can be efficiently derandomized in the online phase. MPC in the preprocessing model aims at increasing the overall efficiency of MPC protocols, but the focus lies on the online phase. The goal here is to satisfy a practical need for fast evaluation of functions once the parties' inputs are known, at the cost of a relatively slower offline phase, which can take place anytime prior to the actual online execution.

In the rest of this chapter we present functionalities and protocols in the preprocessing model, providing security against an active, static adversary in the full-threshold setting (i.e., who can corrupt up to $n-1$ out of the n protocol participants). We assume an additive secret sharing scheme implementing the MPC functionalities, and denote the authenticated sharing of a value x as $[\![x]\!]$. Each party P_i then holds a share $x_i \in \mathbb{F}$, such that $x = \sum_{i=1}^{n} x_i$.

5.1 The Arithmetic Black Box Model

As discussed in Chapter 4, to prove a MPC protocol secure, we first need to model what it is expected to do, using a so-called *functionality*. The functionality models what the trusted party of the ideal world should provide the protocol participants with, based on the requested commands. Here we model the most common functionality for arithmetic MPC over any field $\mathbb{F}$, which is known as the *Arithmetic Black Box*. This functionality is denoted as $\mathcal{F}_{\mathsf{ABB}}$ and it is presented in Figure 5.1. In the functionality, we denote by $\mathcal{S}$ the ideal world adversary.

Functionality $\mathcal{F}_{\mathsf{ABB}}$

Initialize: On input $(\mathsf{Init}, \mathbb{F})$ from all parties store the field $\mathbb{F}$ and initialize a new database, $\mathsf{DB} := \varnothing$.

Input: On input $(\mathsf{Input}, i, \mathsf{id}_1, \ldots, \mathsf{id}_\ell, x_1, \ldots x_\ell, \mathsf{sid})$ from P_i and $(\mathsf{Input}, i, \mathsf{id}_1, \ldots \mathsf{id}_\ell, \perp, \mathsf{sid})$ from all other parties, where $x_i \in \mathbb{F}$ and id_i are distinct new identifiers, append the ℓ entries (id_i, x_i) to the database DB.

Add: On input $(\mathsf{Add}, \mathsf{id}_x, \mathsf{id}_y, \mathsf{id}_z, \mathsf{sid})$ from all parties, where id_x and id_y are identifiers in the database and id_z is a new identifier, retrieve (id_x, x) and (id_y, y) from memory and append the entry $(\mathsf{id}_z, x + y)$ to the database DB.

Multiply: On input $(\mathsf{Multiply}, \mathsf{id}_x, \mathsf{id}_y, \mathsf{id}_z, \mathsf{sid})$ from all parties, where id_x and id_y are identifiers in the database and id_z is a new identifier, retrieve (id_x, x) and (id_y, y) from memory, compute $z := x \cdot y$, and append (id_z, z) to DB.

Output: On input $(\mathsf{Output}, \mathsf{id}_x, \mathsf{sid})$ from all parties, retrieve the entry (id_x, x) from DB, send x to $\mathcal{S}$, and await a message OK or Abort; if the message is OK then send x to all honest parties, and otherwise send the message Abort to all honest parties and halt.

Figure 5.1: Arithmetic Black Box Functionality.

5.2 Active Security

For actively secure MPC protocols, authentication is required. Authentication provides us with a guarantee that the parties cannot change their inputs, or intermediate values needed for the computation, during the protocol execution. To this end information theoretic MACs are being generated (see also Section 3.4.2 for an explanation on MACs). Then, we need to check these MACs to confirm that parties did not cheat during the protocol execution by altering their inputs or providing inconsistent intermediate values during the

computation. To maintain the linearity of the additive secret sharing scheme, so as to be able to evaluate linear functions locally on the shares, without any communication between the parties, the MACs must also be linear. The literature on preprocessing-based MPC offers two ways of deploying MACs for active security: using *pairwise MACs*, where each share is individually MACed, or using *global MACs*, where the actual input values are MACed, and the computed MACs are afterwards also shared.

5.2.1 Pairwise MACs

Following the paradigm of the well-known BeDOZa protocol [9], pairwise MACs can be generated and verified as follows. The MAC key K is a random pair $K = (\alpha, \beta) \in \mathbb{F}^2$. The MAC on a value x is then generated as: $\mathsf{MAC}_K(x) = \alpha x + \beta \in \mathbb{F}$. To apply now this MACing scheme we have one party P_i holding the share x, and the MAC $\mathsf{MAC}_K(x)$, while another party P_j holds K. This way when P_i reveals their share to P_j they can no longer lie about it, as the MAC verification would reveal such a cheating attempt. To preserve linearity of the MACs we fix the α component of the key, so that we can add the MACs and get as the sum a valid MAC on the sum of the underlying shares.

When pairwise MACs are used, a sharing of the secret value x is represented as:

$$[\![x]\!] = \{x_i, \{K^i_{x_j}, \mathsf{MAC}_{K^j_{x_i}}(x_i)\}_{j=1}^n\}_{i=1}^n.$$

Thus, for every share x_i, all parties $P_j \neq P_i$ hold a MAC key $K^i_{x_j}$, while party P_i holds their own share x_i, and a pairwise MAC $\mathsf{MAC}_{K^j_{x_i}}(x_i)$ with each of the other parties P_j. Although effective, pairwise MACs are inefficient, since for every input x we need to generate, exchange, store, and verify $\mathcal{O}(n^2)$ MACs.

5.2.2 Global MACs

Global MACs, first introduced in the original SPDZ protocol [35], overcome the efficiency limitations of pairwise MACs in the following way: instead of many two-component keys $K = (\alpha, \beta) \in \mathbb{F}^2$ (one for each pair of parties), they have one single MAC key $\alpha \in \mathbb{F}$, which is secret shared among the protocol participants. Moreover, instead of MACing every share individually, the actual secret value x is being MACed. The parties also hold a share of the MAC key $\alpha \in \mathbb{F}$, but this is global (i.e., a single MAC key for the whole computation, regardless of the number of inputs, or circuit gates). This results in a sharing of the following form:

$$[\![x]\!] = \{x_i, \mathsf{MAC}_\alpha(x)_i\}_{i=1}^n,$$

where $\mathsf{MAC}_\alpha(x)_i$ is the i^{th} additive share of the MAC on x.

Global MACs, as it can be seen from the sharing representation, immediately reduce the communication, computation, and storage cost compared to the pair-wise MACs from quadratic to linear in the number of parties. To ensure correctness of the MACs during the protocol execution, the MAC key α should remain secret for as long as computations are being performed and parties may change their inputs. SPDZ [35] addresses this challenge by opening the MAC key only during the Output command, when no more computations are going to take place in the protocol, and therefore also restricting the Output command to be invoked only once. The actual checking of the MACs takes place also via the Output command.

5.3 Online Phase

The online phase of MPC protocols is constructed in the $\mathcal{F}_{\mathsf{Prep}}$ hybrid model, meaning that for the online phase execution we need to invoke the $\mathcal{F}_{\mathsf{Prep}}$ functionality, which models the preprocessing phase. We elaborate on the preprocessing phase in Section 5.4. The protocol for the online phase essentially realizes the $\mathcal{F}_{\mathsf{ABB}}$ functionality. Here we focus on the so-called SPDZ family of protocols [9, 35, 81, 48, 66, 67], and detail the online MPC protocol, based on global MACs, in Figure 5.2. Following standard notation, we denote the MAC on a value x as $\gamma(x)$, and the i^{th} share of that MAC as $\gamma_i(x)$. Note that in this protocol the MAC-checking procedure is deferred to the end of the protocol; thus, any intermediate values that get reconstructed from the parties' shares are unauthenticated.

Protocol Π_{Online}

Initialize: The parties invoke $\mathcal{F}_{\mathsf{Prep}}.\mathsf{Init}(\mathbb{F})$, receiving the shared secret key $[\![\alpha]\!]$, where each party P_i receives a share α_i s.t. $\alpha = \sum_{i=1}^{n} \alpha_i$; a number of multiplication triples $([\![a]\!], [\![b]\!], [\![c]\!])$, and mask values $(r, [\![r]\!])$.

Input: To share an input x, party P_i takes an available mask value $(r, [\![r]\!])$, of which only P_i knows the secret value r and does the following:

1. Broadcast $\epsilon \leftarrow x - r$.
2. The parties compute $x_i \leftarrow r_i + \epsilon$.
3. The parties compute the MAC shares $\gamma_i(x) \leftarrow \alpha_i \cdot \epsilon + \gamma_i(r)$.
4. The parties store $[\![x]\!]$ as $\{x_i, \gamma_i(x)\}_{i=1}^{n}$

Add: On input $([\![x]\!], [\![y]\!])$, locally compute $[\![x+y]\!] \leftarrow [\![x]\!] + [\![y]\!]$.

Multiply: On input $([\![x]\!], [\![y]\!])$ the parties do the following:

1. Take one authenticated multiplication triple $([\![a]\!], [\![b]\!], [\![c]\!])$ from the preprocessing, s.t. $c = a \cdot b$, compute $[\![\epsilon]\!] \leftarrow [\![x]\!] - [\![a]\!], [\![\rho]\!] \leftarrow [\![y]\!] - [\![b]\!]$, and open these shares to get ϵ, ρ, respectively.
2. Set $[\![z]\!] \leftarrow [\![c]\!] + \epsilon \cdot [\![b]\!] + \rho \cdot [\![a]\!] + \epsilon \cdot \rho$, as the resulting sharing of the multiplication.

Output: To output a secret $[\![x]\!]$ all previously opened values are being checked:

1. The parties compute a random linear combination of all opened values in the protocol.
2. The parties commit on the MACs of their shares of the linear combination.
3. Then, the sharing of the MAC key $[\![\alpha]\!]$ is opened, and the linear combination is checked over the MACs, using the opened MAC key.
4. If the previous check was successful, all parties broadcast their share x_i, check the open value x as previously, and if this last check fails they Abort; otherwise they output x.

Figure 5.2: Protocol for the online MPC phase.

5.4 Preprocessing Phase

The goal of the preprocessing phase is to increase the efficiency of the online phase, by producing input independent material that can be used in the online phase to randomize the circuit. Concretely, two of the main online operations supported can benefit from this approach: the Input, and the Multiply operations. For the Input operation the preprocessing phase provides the so-called mask values. A mask value is a random value $r \in \mathbb{F}$ additively secret shared among all parties, such that all parties hold a share r_i, and the party P_j who wishes to share a private input also holds the secret value r. For the Multiply operation the preprocessing provides us with the so-called multiplication triples, a technique due to Beaver [3]. A multiplication triple is a set of values $(a, b, c) \in \mathbb{F}^3$ additively secret shared among the protocol participants, such that $c = a \cdot b$. These can be used as shown in the online phase to randomize the real protocol inputs of a multiplication gate, so that privacy is maintained when opening them to compute the sharing of the product. The preprocessing functionality $\mathcal{F}_{\mathsf{Prep}}$ is listed in Figure 5.3.

Functionality $\mathcal{F}_{\mathsf{Prep}}$

Initialize: On input $(\mathsf{Init}, \mathbb{F})$ from all parties store the field $\mathbb{F}$, randomly sample $\alpha_i \in \mathbb{F}, \forall i = 1, \ldots, n$, compute the global MAC key $\alpha := \sum_{i=1}^{n} \alpha_i$ and initialize a new database, $\mathsf{DB} := \varnothing$.

Mask: On input $(\mathsf{Mask}, j, \mathsf{id}_r, \mathsf{sid})$ from all parties, where id_r is a new identifier in the database, call $\mathsf{Sample}(\mathsf{id}_r)$, append (id_r, r) to DB, send r to P_j, and $[\![r]\!]$ to all parties $P_i, i = 1, \ldots, n$.

Triple: On input $(\mathsf{Triple}, \mathsf{id}_a, \mathsf{id}_b, \mathsf{id}_c, \mathsf{sid})$ from all parties, where id_a, id_b, and id_c are new identifiers in the database, call $\mathsf{Sample}(\mathsf{id}_a), \mathsf{Sample}(\mathsf{id}_b)$; compute $c := a \cdot b$, and sample uniformly authenticated shares $[\![c]\!] = (c_i, \gamma_i(c))$, s.t. $c = \sum_{i=1}^{n} c_i$ and $\gamma(c) = \sum_{i=1}^{n} \gamma_i(c)$; append $(\mathsf{id}_a, a), (\mathsf{id}_b, b), (\mathsf{id}_c, c)$ to DB; and send the triple $([\![a]\!], [\![b]\!], [\![c]\!])$ to all parties $P_i, i = 1, \ldots, n$.

Internal Procedure:

Sample: When $(\mathsf{Sample}, \mathsf{id}_x)$ is invoked, randomly sample $x_i \in \mathbb{F}, \forall i = 1, \ldots, n$, compute $x := \sum_{i=1}^{n} x_i$, compute $\gamma(x) := \alpha \cdot x$ and set each MAC share $\gamma_i(x)$ uniformly s.t. $\sum_{i=1}^{n} \gamma_i(x) = \gamma(x)$. Then store the authenticated shares $[\![x]\!] = (x_i, \gamma_i(x))$.

Figure 5.3: Preprocessing Functionality.

To create the multiplication triples in the preprocessing, two approaches have been taken: based on OT [81, 48, 66], and based on SHE [9, 35, 67].

5.4.1 OT-based Preprocessing

OT-based preprocessing is based on Gilboa's multiplication protocol [52], and the primitive known as OPE (Oblivious Polynomial Evaluation), discussed in Section 3.2.5. OT-based preprocessing requires pairwise communication channels between the parties. First each pair of parties needs to engage in a 2PC multiplication protocol, which outputs an encrypted sharing of the product of their inputs. The underlying encryption scheme used in the OT protocol must be linearly homomorphic to allow the evaluation in a single communication round. This extends from the two-party to the multiparty setting, by having each ordered pair of parties invoking the abovementioned 2PC multiplication protocol. Given the pairwise communication channels requirement, this approach may be more suitable for two-party computation, but considering MPC, it does not scale well in the number of parties.

5.4.2 SHE-based Preprocessing

SHE-based preprocessing has more stringent requirements for the encryption scheme. Concretely, a linear homomorphic encryption scheme does not suffice, but it can be implemented with a depth-1 SHE scheme (i.e., the SHE scheme must be parametrized to correctly allow the computation of one secret-secret multiplication). Using the depth-1 SHE scheme, we can now compute the multiplication on the encrypted values, and then perform distributed decryption, so that each party obtains a share of the desired product. This is done without deploying OT, which eliminates the need for pairwise communication channels, and allows us to use a broadcast channel among the parties. Hence, SHE-based preprocessing is more efficient for MPC applications, especially as the number of parties n grows.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In the last decade MPC started turning from a theoretical concept into practical solutions. Nowadays MPC protocols have expanded from mere theoretical constructs, or proof-of-concept implementations to actual company services deployed on a wide range of privacy-preserving applications [32, 95, 89]. Although we can argue that the field is reaching maturity, there is room for improvement both on the technical, or scientific side, as well as on the societal acceptance of the technologies in question.

As discussed in Chapter 1, there are two ways one can approach the research challenges and devise solutions in MPC: either look at a particular problem, and tailor the MPC solution fully to the setting, so as to get optimal results for the scenario at hand, or aim at improving a particular MPC primitive, which is of more general interest and can be applied on different application scenarios. Either way, the results are beneficial to the research community and the general development of the field. Interestingly, one may begin a research project focusing on a particular application scenario, and by digging deeper into the details of addressing the problem one may still invent a primitive of more general interest. Vice versa, one may attempt to invent a new primitive, and instead end up with a construction that only serves particular application scenarios.

We now take a closer look at the conclusions drawn by the concrete contributions of this thesis. Our work on privacy-preserving genome-wide association studies, showed that secure computation, and in particular both HE and MPC are

mature enough to be deployed on large-scale applications, with realistic, stringent security requirements, and at the same time realistic scalability. The comparison between the two approaches confirmed that MPC outperforms HE, despite certain optimizations that we adopted for the secure HE comparisons –a crucial component of the protocol in terms of computational cost. However, while total execution time of the protocol is of utmost importance, we should not underestimate the significance of the commmunication cost: while HE does not require any communication during the protocol execution, MPC requires the parties to synchronously and interactively execute the protocol. Moreover, the security guarantees that the two approaches may offer are different: while HE assumes that the computation party is semi-honest, MPC can offer active security guarantees, which we opted for in this particular application.

When considering interdisciplinary research, where secure computation is only one of the components that affect the effectiveness and efficiency of a solution, leveraging the advances of the complementary research disciplines can improve secure computation solutions. Our image classification protocol, although simple, outperformed the state-of-the-art privacy-preserving classification techniques both in terms of efficiency, and in terms of classification accuracy. While the related work focused on producing secure classification algorithms equivalent to the ones developed by the machine learning community, we approached the problem differently by looking at how can we achieve the same accuracy results, efficiently in the privacy-preserving domain. In this attempt we discovered that we could leverage the techniques of transfer learning, to prepare our private data in the plaintext domain, without compromising privacy. The related work, on the other hand, focused on closely resembling the neural network techniques used in the plaintext domain, but translated into secure protocols. This way they ended up implementing complex, and therefore also less efficient functions in the privacy-preserving domain.

Many works in secure computation have shown that particular approaches, especially related to data representations execute some tasks efficiently, while they perform poorly at other tasks. Specifically, Boolean circuits are more suitable for non-linear operations, while arithmetic circuits are significantly more efficient than Boolean circuits when it comes to linear operations. Given that most modern applications require the execution of a combination of linear and non-linear tasks, finding ways to bridge these two representations is imperative. Our work on multiparty arithmetic garbling indeed makes a contribution towards this end, by proposing a design for a garbled selector gate –more efficient than the one proposed in prior work– which allows this interfacing between arithmetic and Boolean gates.

6.2 Future Work

In this section we discuss possible future work that would be a useful follow up for our research.

Switching protocols. Although switching protocols, and frameworks to combine different secure computation methods already exist, this particular subfield is young and it requires further exploration. A few years ago, one of the first encryption switching protocols appeared [29], consisting of a MPC protocol that facilitates switching between an additively homomorphic and a multiplicatively homomorphic encryption scheme. In this thesis, we have concluded that switching between arithmetic and Boolean representations is needed for modern secure applications. Although we proposed a particular garbled gate, namely a selector gate, serving as an interface between Boolean and arithmetic representations, a seamless and efficient conversion between the two representations remains an interesting open problem. We have also discussed the already existing seamless combination of SHE with MPC, where SHE serves as the main tool to create preprocessing material for MPC protocols. Moreover, there exist secure computation frameworks where secret-sharing based and garbling-based sub-protocols can be combined [39]. Hence, the combination possibilities between different representations, or different computation methods are numerous. It remains interesting to study how one computation, or representation method can benefit from the other, and how they can be combined to create efficient secure computation protocols.

Secure computation for non-linear functions. While in certain application scenarios computing a simple statistic, represented by a low-degree polynomial, may be sufficient (e.g., the χ^2 statistic in the genomics application we considered in this thesis), the need to securely evalutate more complex functions increases. The readiness of secure computation to be deployed for computations on sensitive data needed by the scientific community (e.g., machine learning, epidemiology, statistics, genomics, etc.), highly depends on whether we can efficiently compute non-linear functions in the privacy-preserving domain. It is therefore an interesting open problem to improve the current MPC protocols in terms of their ability to deal with non-linear functions efficiently. For instance, one can consider as a stepping stone a heavier preprocessing phase, that can be executed well ahead of time, to facilitate an efficient online computation (e.g., preprocessed look-up tables have been already proposed in the literature [34, 64]).

Preprocessing material for MPC protocols with flexible number of communication rounds. Currently, we can either opt for MPC protocols with constant number of rounds, following the garbling paradigm, or we can opt for protocols, where the number of rounds is linear in the depth of the circuit to

be computed, following the secret-sharing paradigm. This leads to the general rule of selecting one of the two options based on the network over which our MPC protocols will be executed: when we have a high-latency network (e.g., a Wide Area Network - WAN) then GC may be preferred; when we have a low-latency network (e.g., a Local Area Network - LAN) then secret-sharing is preferred. One reason why we are faced with this binary decision is that the preprocessing phase of secret-sharing based MPC creates material for treating at most quadratic functions in the online phase. It is therefore inevitable for the number of communication rounds to depend linearly in the depth of the circuit to be computed. Hence, an interesting research direction is to explore how we can devise flexible MPC protocols anywhere between the two extremes of the spectrum. Creating preprocessing material for higher degree functions, will allow one to decide to trade additional preprocessing time for a more efficient online phase that completes its execution in the desired number of communication rounds.

MPC for machine learning applications. Privacy-preserving machine learning (or artificial intelligence - AI) is currently on the rise, as the societal need for them is becoming more and more urgent. For example, we are surrounded by smart applications and IoT devices, all of which make decisions based on some kind of machine learning algorithm. Given the vast data collection that these applications require for their training phase, but also the potentially sensitive conclusions that they draw during classification, privacy is an essential consideration. Secure machine learning offers a wide range of applications, as the security requirements differ from setting to setting, and there are different protocols to be securely implemented: protocols for the training phase, or protocols for the classification phase. As we concluded also in this thesis, efficient and accurate solutions can be constructed by leveraging the advances in both the MPC field and the machine learning field, and wisely combining them together. Exploring this area of privacy-preserving AI is indeed an interesting future direction.

It is the most interesting to consider what we can still do to improve and advance the field of secure computation. However, we should also recognize the limitations of our research area, and we should make sure to clearly communicate those to any parties interested in this field. The two most profound limitations of MPC, which are also inherent are the following:

1. MPC protects the private inputs of the involved parties, and guarantees that nothing can be learned by observing the execution of the MPC protocol, but as the output gets revealed (publicly, or to a subset of the MPC participants), whatever can be inferred from this output cannot be protected. In fact, the MPC parties may infer more information than

external observers, as they can combine their knowledge of the output, with their knowledge of their own private inputs.

2. MPC allows the protocol participants to input their private values. Even when considering actively secure MPC protocols, which check whether a party tries to cheat, MPC does not provide any mechanism to force the participants to input their "true" private values. MPC can only check whether these inputs are being consistently used throughout the protocol, but it cannot enforce honesty as far as the parties' inputs are concerned.

Bibliography

[1] Albrecht, M., Bai, S., and Ducas, L. A Subfield Lattice Attack on Overstretched NTRU Assumptions. In *Annual International Cryptology Conference* (2016), Springer, pp. 153–178.

[2] Ball, M., Malkin, T., and Rosulek, M. Garbling Gadgets for Boolean and Arithmetic Circuits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), pp. 565–577.

[3] Beaver, D. Efficient Multiparty Protocols Using Circuit Randomization. In *Annual International Cryptology Conference* (1991), Springer, pp. 420–432.

[4] Beaver, D. Correlated Pseudorandomness and the Complexity of Private Computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (1996), pp. 479–488.

[5] Beaver, D., Micali, S., and Rogaway, P. The Round Complexity of Secure Protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing* (1990), pp. 503–513.

[6] Beimel, A. Secret-Sharing Schemes: A Survey. In *International Conference on Coding and Cryptology* (2011), Springer, pp. 11–46.

[7] Ben-Efraim, A. On Multiparty Garbling of Arithmetic Circuits. In *International Conference on the Theory and Application of Cryptology and Information Security* (2018), Springer, pp. 3–33.

[8] Benaloh, J. Dense Probabilistic Encryption. In *Proceedings of the workshop on selected areas of cryptography* (1994), pp. 120–128.

[9] Bendlin, R., Damgård, I., Orlandi, C., and Zakarias, S. Semi-Homomorphic Encryption and Multiparty Computation. In *Annual*

International Conference on the Theory and Applications of Cryptographic Techniques (2011), Springer, pp. 169–188.

[10] Blakley, G. R. Safeguarding Cryptographic Keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)* (1979), IEEE, pp. 313–318.

[11] Bogetoft, P., Christensen, D. L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J. D., Nielsen, J. B., Nielsen, K., Pagter, J., Schwartzbach, M., and Toft, T. Secure Multiparty Computation Goes Live. In *International Conference on Financial Cryptography and Data Security* (2009), Springer, pp. 325–343.

[12] Boneh, D., Goh, E.-J., and Nissim, K. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of cryptography conference* (2005), Springer, pp. 325–341.

[13] Bonte, C., Makri, E., Ardeshirdavani, A., Simm, J., Moreau, Y., and Vercauteren, F. Towards Practical Privacy-Preserving Genome-Wide Association Study. *BMC bioinformatics 19*, 1 (2018), 1–12.

[14] Brakerski, Z. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Annual Cryptology Conference* (2012), Springer, pp. 868–886.

[15] Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computation Theory (TOCT) 6*, 3 (2014), 1–36.

[16] Brakerski, Z., and Vaikuntanathan, V. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *Annual cryptology conference* (2011), Springer, pp. 505–524.

[17] Brakerski, Z., and Vaikuntanathan, V. Efficient Fully Homomorphic Encryption from (Standard) LWE. *SIAM Journal on Computing 43*, 2 (2014), 831–871.

[18] Brassard, G., Chaum, D., and Crépeau, C. Minimum Disclosure Proofs of Knowledge. *Journal of computer and system sciences 37*, 2 (1988), 156–189.

[19] Burgess, M. How to Check if your LinkedIn Account was Hacked. `https://www.wired.co.uk/article/linkedin-data-breach-find-out-included`. Accessed: 02/10/2020.

[20] Campbell, M., Egerstedt, M., How, J. P., and Murray, R. M. Autonomous Driving in Urban Environments: Approaches, Lessons and Challenges. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368*, 1928 (2010), 4649–4672.

[21] Canetti, R. Security and Composition of Multiparty Cryptographic Protocols. *Journal of CRYPTOLOGY 13*, 1 (2000), 143–202.

[22] Canetti, R. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science* (2001), IEEE, pp. 136–145.

[23] Chaum, D., Crépeau, C., and Damgard, I. Multiparty Unconditionally Secure Protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing* (1988), pp. 11–19.

[24] Chen, M., Cohen, R., Doerner, J., Kondi, Y., Lee, E., Rosefield, S., and Shelat, A. Multiparty Generation of an RSA Modulus. In *Annual International Cryptology Conference* (2020), Springer, pp. 64–93.

[25] Cheon, J. H., Kim, A., Kim, M., and Song, Y. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *International Conference on the Theory and Application of Cryptology and Information Security* (2017), Springer, pp. 409–437.

[26] Chillotti, I., Gama, N., Georgieva, M., and Izabachene, M. Faster Fully Homomorphic Encryption: Bootstrapping in Less than 0.1 Seconds. In *international conference on the theory and application of cryptology and information security* (2016), Springer, pp. 3–33.

[27] Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)* (1985), IEEE, pp. 383–395.

[28] Cook, D. J., Augusto, J. C., and Jakkula, V. R. Ambient Intelligence: Technologies, Applications, and Opportunities. *Pervasive and Mobile Computing 5*, 4 (2009), 277–298.

[29] Couteau, G., Peters, T., and Pointcheval, D. Encryption Switching Protocols. In *Annual International Cryptology Conference* (2016), Springer, pp. 308–338.

[30] Cramer, R., Damgård, I., and Maurer, U. General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme. In

International Conference on the Theory and Applications of Cryptographic Techniques (2000), Springer, pp. 316–334.

[31] Cramer, R., Gennaro, R., and Schoenmakers, B. A Secure and Optimally Efficient Multi-Authority Election Scheme. *European transactions on Telecommunications 8*, 5 (1997), 481–490.

[32] Cybernetica. `https://cyber.ee/`, 2020.

[33] Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., and Smart, N. P. Practical Covertly Secure MPC for Dishonest Majority– or: Breaking the SPDZ Limits. In *European Symposium on Research in Computer Security* (2013), Springer, pp. 1–18.

[34] Damgård, I., Nielsen, J. B., Nielsen, M., and Ranellucci, S. The TinyTable Protocol for 2-Party Secure Computation, or: Gate-Scrambling Revisited. In *Annual International Cryptology Conference* (2017), Springer, pp. 167–187.

[35] Damgård, I., Pastro, V., Smart, N., and Zakarias, S. Multiparty Computation from Somewhat Homomorphic Encryption. In *Annual Cryptology Conference* (2012), Springer, pp. 643–662.

[36] Data61. MP-SPDZ: A Versatile Framework for Multi-Party Computation. `https://github.com/data61/MP-SPDZ`, 2019.

[37] de Hoogh, S. J. A. *Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming.* PhD thesis, Technische Universiteit Eindhoven, 2012.

[38] Delpech de Saint Guilhem, C., Makri, E., Rotaru, D., and Tanguy, T. The return of Eratosthenes: Secure Generation of RSA Moduli using Distributed Sieving. Cryptology ePrint Archive, Report 2021/565, 2021.

[39] Demmler, D., Schneider, T., and Zohner, M. ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS* (2015).

[40] Dinur, I., and Nissim, K. Revealing Information while Preserving Privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2003), pp. 202–210.

[41] Duhigg, C. How Companies Learn Your Secrets. `https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html`. Accessed: 02/10/2020.

[42] Dwork, C. Differential Privacy: A Survey of Results. In *International conference on theory and applications of models of computation* (2008), Springer, pp. 1–19.

[43] Dwork, C., McSherry, F., Nissim, K., and Smith, A. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of cryptography conference* (2006), Springer, pp. 265–284.

[44] ElGamal, T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE transactions on information theory 31*, 4 (1985), 469–472.

[45] Escudero, D., Ghosh, S., Keller, M., Rachuri, R., and Scholl, P. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. In *International Conference on the Theory and Application of Cryptology and Information Security* (2020), Springer, pp. 3–33.

[46] Even, S., Goldreich, O., and Lempel, A. A Randomized Protocol for Signing Contracts. *Communications of the ACM 28*, 6 (1985), 637–647.

[47] Fan, J., and Vercauteren, F. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[48] Frederiksen, T. K., Keller, M., Orsini, E., and Scholl, P. A Unified Approach to MPC with Preprocessing Using OT. In *International Conference on the Theory and Application of Cryptology and Information Security* (2015), Springer, pp. 711–735.

[49] Gennaro, R., Rabin, M. O., and Rabin, T. Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing* (1998), pp. 101–111.

[50] Gentry, C. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing* (2009), pp. 169–178.

[51] Gentry, C., and Halevi, S. Implementing Gentry's Fully-Homomorphic Encryption Scheme. In *Annual international conference on the theory and applications of cryptographic techniques* (2011), Springer, pp. 129–148.

[52] Gilboa, N. Two Party RSA Key Generation. In *Annual International Cryptology Conference* (1999), Springer, pp. 116–129.

[53] Goldreich, O. *Secure Multi-Party Computation*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1998.

[54] Goldreich, O. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge university press, 2009.

[55] Goldreich, O., Micali, S., and Wigderson, A. How to Play Any Mental Game, or a Completeness Theorem for Protocols with Honest Majority. *Proc. the Nineteenth Annual ACM Symposium on the Theory of Computing* (1987), 218–229.

[56] Goldwasser, S., and Micali, S. Probabilistic Encryption & How to Play Mental Poker Keeping Secret all Partial Information. In *14th ACM Symposium on the Theory of Computing* (1982), vol. 365.

[57] Goldwasser, S., Micali, S., and Rackoff, C. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on computing 18*, 1 (1989), 186–208.

[58] Hackett, R. What to Know about the Ashley Madison Hack. `https://fortune.com/2015/08/26/ashley-madison-hack/`. Accessed: 02/10/2020.

[59] Homer, N., Szelinger, S., Redman, M., Duggan, D., Tembe, W., Muehling, J., Pearson, J. V., Stephan, D. A., Nelson, S. F., and Craig, D. W. Resolving Individuals Contributing Trace Amounts of DNA to Highly Complex Mixtures using High-Density SNP Genotyping Microarrays. *PLoS Genet 4*, 8 (2008), e1000167.

[60] Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. Extending Oblivious Transfers Efficiently. In *Annual International Cryptology Conference* (2003), Springer, pp. 145–161.

[61] Ishai, Y., and Kushilevitz, E. Private Simultaneous Messages Protocols with Applications. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems* (1997), IEEE, pp. 174–183.

[62] Jeckmans, A. J. P. *Cryptographically-Enhanced Privacy for Recommender Systems*. PhD thesis, University of Twente, the Netherlands, 2014.

[63] Keller, M. MP-SPDZ: A Versatile Framework for Multi-Party Computation. Cryptology ePrint Archive, Report 2020/521, 2020.

[64] Keller, M., Orsini, E., Rotaru, D., Scholl, P., Soria-Vazquez, E., and Vivek, S. Faster Secure Multi-Party Computation of AES and DES Using Lookup Tables. In *International Conference on Applied Cryptography and Network Security* (2017), Springer, pp. 229–249.

[65] Keller, M., Orsini, E., and Scholl, P. Actively Secure OT Extension with Optimal Overhead. In *Annual Cryptology Conference* (2015), Springer, pp. 724–741.

[66] Keller, M., Orsini, E., and Scholl, P. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), pp. 830–842.

[67] Keller, M., Pastro, V., and Rotaru, D. Overdrive: Making SPDZ Great Again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2018), Springer, pp. 158–189.

[68] Kilian, J. Founding Cryptography on Oblivious Transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing* (1988), pp. 20–31.

[69] Kitchin, R. The Real-Time City? Big Data and Smart Urbanism. *GeoJournal 79*, 1 (2014), 1–14.

[70] Kolesnikov, V., Mohassel, P., and Rosulek, M. FleXOR: Flexible Garbling for XOR Gates that Beats Free-XOR. In *Annual Cryptology Conference* (2014), Springer, pp. 440–457.

[71] Kolesnikov, V., and Schneider, T. Improved Garbled Circuit: Free XOR Gates and Applications. In *International Colloquium on Automata, Languages, and Programming* (2008), Springer, pp. 486–498.

[72] Krumholz, H. M. Big Data and New Knowledge in Medicine: The Thinking, Training, and Tools Needed for a Learning Health System. *Health Affairs 33*, 7 (2014), 1163–1170.

[73] López-Alt, A., Tromer, E., and Vaikuntanathan, V. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing* (2012), pp. 1219–1234.

[74] Lupton, D. The Commodification of Patient Opinion: The Digital Patient Experience Economy in the Age of Big Data. *Sociology of health & illness 36*, 6 (2014), 856–869.

[75] Makri, E., Rotaru, D., Smart, N. P., and Vercauteren, F. EPIC: efficient private image classification (or: Learning from the masters). In *Cryptographers' Track at the RSA Conference* (2019), Springer, pp. 473–492.

[76] Makri, E., Rotaru, D., Vercauteren, F., and Wagh, S. Rabbit: Efficient Comparison for Secure Multi-Party Computation. In *International Conference on Financial Cryptography and Data Security* (2021), Springer. *To appear.*

[77] Makri, E., and Wood, T. Full-Threshold Actively-Secure Multiparty Arithmetic Circuit Garbling. Cryptology ePrint Archive, Report 2019/1098, 2019.

[78] Naor, M., and Pinkas, B. Oblivious Transfer and Polynomial Evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing* (1999), pp. 245–254.

[79] Naor, M., Pinkas, B., and Sumner, R. Privacy Preserving Auctions and Mechanism Design. In *Proceedings of the 1st ACM conference on Electronic commerce* (1999), pp. 129–139.

[80] Narayanan, A., and Shmatikov, V. Robust De-anonymization of Large Sparse Datasets. In *2008 IEEE Symposium on Security and Privacy (S & P)* (2008), IEEE, pp. 111–125.

[81] Nielsen, J. B., Nordholt, P. S., Orlandi, C., and Burra, S. S. A New Approach to Practical Active-Secure Two-Party Computation. In *Annual Cryptology Conference* (2012), Springer, pp. 681–700.

[82] Paillier, P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *International conference on the theory and applications of cryptographic techniques* (1999), Springer, pp. 223–238.

[83] Pinkas, B., Schneider, T., Smart, N. P., and Williams, S. C. Secure Two-Party Computation is Practical. In *International conference on the theory and application of cryptology and information security* (2009), Springer, pp. 250–267.

[84] Rabin, M. O. How to Exchange Secrets by Oblivious Transfer. Technical Memo TR-81, 1981.

[85] Regev, O. On Lattices, Learning With Errors, Random Linear Codes, and Cryptography. *Journal of the ACM (JACM) 56*, 6 (2009), 1–40.

[86] Rivest, R. L., Adleman, L., and Dertouzos, M. L. On Data Banks and Privacy Homomorphisms. *Foundations of secure computation 4*, 11 (1978), 169–180.

[87] Rivest, R. L., Shamir, A., and Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–126.

[88] Schoenmakers, B. A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting. In *Annual International Cryptology Conference* (1999), Springer, pp. 148–164.

[89] Sepior. `https://sepior.com/`, 2020.

[90] Shamir, A. How to Share a Secret. *Communications of the ACM 22*, 11 (1979), 612–613.

[91] Shannon, C. E. Communication Theory of Secrecy Systems. *The Bell system technical journal 28*, 4 (1949), 656–715.

[92] Smart, N. P., and Vercauteren, F. Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In *International Workshop on Public Key Cryptography* (2010), Springer, pp. 420–443.

[93] Stottelaar, B., Senden, J., and Montoya, L. Online Social Sports Networks as Crime Facilitators. *Crime science 3*, 1 (2014), 8.

[94] Sweeney, L. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10*, 05 (2002), 557–570.

[95] Unbound. `https://www.unboundtech.com/`, 2020.

[96] Van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V. Fully Homomorphic Encryption over the Integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2010), Springer, pp. 24–43.

[97] Wong, J. C. The Cambridge Analytica Scandal Changed the World – But it Didn't Change Facebook. `https://www.theguardian.com/technology/2019/mar/17/the-cambridge-analytica-scandal-changed-the-world-but-it-didnt-change-facebook`. Accessed: 02/10/2020.

[98] Yao, A. C. Protocols for Secure Computations. In *23rd annual symposium on foundations of computer science (FOCS)* (1982), IEEE, pp. 160–164.

[99] Yao, A. C.-C. How to Generate and Exchange Secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)* (1986), IEEE, pp. 162–167.

[100] Zahur, S., Rosulek, M., and Evans, D. Two Halves Make a Whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2015), Springer, pp. 220–250.

Part II

Publications

Chapter 7

Towards Practical Privacy-Preserving Genome-Wide Association Study

Publication data

C. Bonte, E. Makri, A. Ardeshirdavani, J. Simm, Y. Moreau, F. Vercauteren. "Towards practical privacy-preserving genome-wide association study" In *BMC Bioinformatics 19*, no. 1 (2018): 1-12.

This is the extended version of the paper published in BMC Bioinformatics, available online at: `https://eprint.iacr.org/2017/955`.

Towards Practical Privacy-Preserving Genome-Wide Association Study

Charlotte Bonte[2,4], Eleftheria Makri[2,3,4], Amin Ardeshirdavani[1], Jaak Simm[1], Yves Moreau[1,5], Frederik Vercauteren[2,5]

[1] STADIUS KU Leuven
[2] imec-Cosic, Dept. Electrical Engineering, KU Leuven
[3] ABRR Saxion University of Applied Sciences
[4] Joint first authors
[5] Joint last authors

Abstract. The deployment of Genome-wide association studies (GWASs) requires genomic information of a large population to produce reliable results. This raises significant privacy concerns, making people hesitate to contribute their genetic information to such studies. We propose two provably secure solutions to address this challenge: (1) a somewhat homomorphic encryption (HE) approach, and (2) a secure multiparty computation (MPC) approach. Unlike previous work, our approach does not rely on adding noise to the input data, nor does it reveal any information about the patients. Our protocols aim to prevent data breaches by calculating the χ^2 statistic in a privacy-preserving manner, without revealing any information other than whether the statistic is significant or not. Specifically, our protocols compute the χ^2 statistic, but only return a yes/no answer, indicating significance. By not revealing the statistic value itself but only the significance, our approach thwarts attacks exploiting statistic values. We significantly increased the efficiency of our HE protocols by introducing a new masking technique to perform the secure comparison that is necessary for determining significance. We show that full-scale privacy-preserving GWAS is practical, as long as the statistics can be computed by low degree polynomials. Our implementations demonstrated that both approaches are efficient. The secure multiparty computation technique completes its execution in approximately 2 ms for data contributed by one million subjects.

1 Introduction

The goal of a genome-wide association study (GWAS) is to identify genetic variants that are associated with traits. Large-scale sequencing provides reliable information on single nucleotide variants (SNVs). To date, researchers worked mostly on identifying genetic alterations which lead to classification of SNVs and SNPs (single nucleotide polymorphims). Therefore, when we mention SNVs we refer to both frequent SNPs, and less frequent SNVs. A common approach is

to divide the population into a disease, and a healthy group based on whether the individual has the particular disease. Each individual gives a sample DNA from which millions of genetic variants (i.e., SNVs) are identified. If a variant is more frequent in individuals with the disease, it will likely be associated with the specific genetic disorder and be classified as a potential marker of the disease.

1.1 Motivation for the distributed setup with secure computations

Having a large population size is crucial for GWAS, because it allows to improve the accuracy of identified associations, especially for *rare* genetic disorders. Two recent developments result in a significant increase of the available data for GWAS: First, the development of cheap next generation sequencing (NGS). Second, the creation of *distributed genomic databases*, which enable pooling of data from many hospitals, and research centers, further increasing the population sizes of the studies by 10-50 times. Several such distributed databases have recently been proposed, including NGS-Logistics [4], Elixir, and GA4GH Beacon [39].

In studies like GWAS, which use personally identifiable genetic markers of the participants as input, the privacy of the patients and protection of their sensitive data becomes of great importance. It has been shown by Malin et al. [33], that releasing the raw data even after removal of explicit identifiers, does not protect an individual from getting identified. The classical approach to solve this privacy problem involves a trusted third party who first collects both the SNV, and the trait data, then carries out the statistical test, and finally either a) only reveals the very few SNVs that have statistically significant association or b) reveals all aggregate data on SNVs but masks them with sufficient noise to guarantee *differential privacy*. For example, previous works by Uhlerop et al. [40], and by Simmons and Berger [38] have focused on computing a differentially private χ^2 test. However, setting up such a trusted third party has significant legal, and technical difficulties given the sensitive nature of the underlying data.

The aforementioned privacy concerns make both individuals and medical centers hesitant to share this private data. Hence, centralized (third party) datasets collected for research purposes remain small. Our goal is to address this challenge, in a way that the data can be shared without trusting an external third party. In our setup, the medical centers aggregate and encrypt or secret share the patient data before sending it to a third party for research purposes. This ensures the privacy of the input data, because the only party with access to the raw input data is the medical center which gathers it. Hence, our distributed solution allows to combine input data from different medical centers to construct a large dataset for research, while eliminating the privacy implications. Our solution can even scale up to *millions of patients*, and perform *millions or tens of millions of hypothesis tests* per day. This enables the first step towards

large-scale distributed GWAS, where multiple medical centers contribute data, without relying on a trusted third party. Such large data collections would also allow association studies on rare diseases.

Another reason to opt for our secure computation solution instead of the trusted third party one, is that the latter does not provide defense against malicious agents or operating system bugs, which might result in leakage of information. In our case, such a mishap would reveal encrypted values (or shares of a value, resp.), which essentially provides no information to the adversary, as long as the secret key is not comprimised (or the adversary has fewer than n shares, resp.).

1.2 Motivation for the yes/no response

Studies related to GWAS raised even more privacey concerns. Research has brought to light that releasing aggregated statistics related to GWASs leaks information in an implicit way. Therefore, it is not enough to protect only the input data; care has to be taken when releasing aggregated results to the public, as well. The work of Homer et al. [23] showed that the presence of an individual in the case group can be determined from the aggregated allele frequencies. One can argue that this attack requires an adversary to have at least $10,000$ SNVs from the victim. However, we assume that with the current sequencing techniques, this is no longer a challenge and hence Homer's attack is posing a real threat nowadays. By computing with encrypted or secret shared data, and only revealing a boolean value indicating significance, we prevent adversaries from obtaining the aggregated allele frequencies, thus protecting against Homer's attack.

Shortly following Homer's attack, Wang et al. [41] reported an attack based on statistical values reported in GWAS papers. Even though, the attack of Wang et al. [41] requires more statistical data than what our solution would reveal, such developments show that we need to be careful with the amount of information we publish. Our solution anticipates future statistical attacks, by not publishing any statistic values at all.

1.3 Additional properties of the our setup

Our proposal consists of a cryptographic approach, where the trusted third party performing research is replaced by a *privacy-preserving* system, which receives the input in encrypted (protected) form from a set of distributed parties (e.g., hospitals), performs the χ^2 test, and only publicly discloses whether the current test is significant or not. Since nothing except the final answer is revealed during

the execution of our protocols, the proposed system enjoys various security guarantees, even against malicious agents who gain access to the servers executing the system.

Even though the aforementioned attacks show it is not a good idea to reveal the χ^2 value, the value itself would be highly interesting for research purposes. Therefore, it is worth mentioning that our current system can be easily adapted to return the significance value itself. However, since revealing the values can cause privacy issues, we suggest to incorperate an authentication process to the system if the χ^2 value should be revealed. This way the access to the actual χ^2 values can be restricted to authenticated users. As such, the researchers can have access to the actual result, while it stays hidden from the public and therefore cannot be abused in an attack like the aforementioned ones.

By only revealing the yes/no answer, our system indicates whether the SNV is a possible marker. To determine whether or not this SNV is actually causally linked to the disease more statistics need to be computed. Therefore, we assume that for the selected SNVs –indicated by our system– the researcher would request specific patient data from the different centers for further analysis. We assume this will happen with the current techniques for requesting data for GWASs. However, we expect patients to be more inclined to share their data for research, even despite the potential privacy concerns, when the researchers explain to them, with the aid of the public tables, that their data is highly relevant for the study of a specific disease.

Additionally, it is common practice in GWASs, and more general bioinformatics studies to publish only when significant results are found. This means that all the insignificant (yet identified) results are not published, despite the fact that they could also contribute in finding, or eliminating interesting correlations. In fact, a non-significant correlation between a genotype and a phenotype can serve as a proof that a certain mutation is not related to a disease. Our solution comes to bridge this gap, as we aim to construct a public table, listing all possible mutations, versus all possible phenotypes, and indicating whether the initial relationship between them (indicated by the χ^2 test) is significant or not. By publishing also the insignificant results in our public table, mutations not related to phenotypes can be immediately shown, allowing the researchers to discard them, and focus only on the significant ones.

To allow for a privacy-preserving system addressing our challenges, we propose two secure approaches: one based on homomorphic encryption (HE), and one based on multiparty computation (MPC). We also compare their security guarantees, and their efficiency in terms of execution time of practical implementations. Homomorphic encryption refers to a set of cryptographic tools that allow certain computations to take place in the encrypted domain, while the re-

sulting ciphertext, when decrypted, is the expected (correct) result of operations on the plaintext data. Secure multiparty computation aims at allowing a similar functionality, amongst several mutually distrusting parties, who wish to compute a function without revealing their private inputs. With the latter approach, communication between the computing parties is required for the execution of the cryptographic protocols.

In the MPC setting, there are two main security models used, offering passive, or active security, respectively. *Passive security*, also known as security in the semi-honest model, assumes that the protocol participants are honest-but-curious. This means that they are trying to collect as much information as possible from the protocol execution, but they do follow the protocol instructions honestly. *Active security*, also known as malicious security, offers stronger security guarantees, assuming that adversaries or corrupted protocol participants may arbitrarily deviate from the protocol instructions. In both security models, we can build protocols assuming an honest majority of the protocol participants, or a dishonest majority. Our solution with MPC offers the highest security guarantees being built in the malicious model, with dishonest majority.

Specifically, we make the following contributions:

- We propose the first *somewhat homomorphic encryption* approach to withstand GWAS attacks such as the ones described by Homer et al. [23].
- We develop a *multiparty computation* solution for GWAS that is efficient for realistic sample sizes.
- We propose a new masking technique to allow efficient secure comparisons.
- We compare the security, and efficiency of HE and MPC on a real-life application.
- We demonstrate the practicality of our solutions, based on their short running times, which are in the range of 1.9-2.4 ms for the MPC approach.
- We show that our solution scales logarithmically in the number of subjects contributing their genetic information, allowing us to treat current population sizes, and being able to scale to larger (future) GWASs for millions of people.

2 Related Work

2.1 Homomorphic encryption approach

There has already been some work on using homomorphic encryption to preserve the privacy of the patients while performing statistics on genome data. Kim et

al. [28] present the computation of minor allele frequencies, and the χ^2 statistic with the use of the homomorphic BGV and YASHE encryption schemes. They use a specific encoding technique to improve on the work of Lauter et al. [29]. However, they only compute the allele counts homomorphically, and execute the other operations on the decrypted data. Another work on GWASs using fully homomorphic encryption was published by Lu et al. [31]. They also start from encrypted genotype/phenotype information that is uploaded to a cloud for each person separately. Then they perform the minimal operations necessary to provide someone with access to the decryption key with the necessary values to construct the contingency table for the requested case based on the data present on the cloud. Hence, when performing a request, the scientist gets three encrypted values, and based on those he can, after decryption, reconstruct the contingency table, and compute the χ^2 statistic in the clear. These solutions are not resistant to attacks like the one described by Homer et al. [23]. Our solution improves on these previous works by performing the χ^2 computation in the encrypted domain, and revealing only whether or not the χ^2 value is significant for this case, which makes the previously mentioned attacks impossible.
Sadat et al. [37] propose a hybrid system called SAFETY, to compute various statistical values over genomic data. This hybrid system consists of a combination of the partially homomorphic Paillier scheme with the secure hardware component of Intel Software Guard Extensions (Intel SGX) to ensure both high efficiency, and privacy. With this hybrid system they propose a more efficient way to get the total counts of all patients for a specific case. By using the additive property of the homomorphic Paillier scheme, they reduce the computational overhead of decrypting all individual encrypted outputs received from the different servers. Afterwards it uses the Intel SGX component to perform the χ^2 computations. Even though, the results of this system scale well for increasing number of servers that provide data for the computation, the system does not provide the same functionality as our solution. Sadat et al. [37] mention that the only privacy guarantee for the final computation result against the attack described by Homer et al. [23] is the assumption that the researcher decrypting the result is semi-honest. This is the main difference with our work: with our solution only the significance of the test will be made public. As mentioned before, the current system can be easily adapted to return the χ^2 value itself but due to known attacks we want to avoid making these values public. Hence, we believe that if our system is adapted to reveal the χ^2 values, it should only reveal these values after authentication of the requesting party.
Zhang et al. [43], construct an algorithm, which performs the whole χ^2 statistic in the homomorphic domain. To compute the division, they construct a lookup table in which they link the result of their computation with the nominator and

denominator of the corresponding, simplified fraction. Therefore, an authenticated user can look up the correct fraction in the lookup table after decrypting the result, and hence recover the result of the χ^2 statistic. Even though their strategy performs well, it does not scale enough to treat the large datasets we envision in our application. Increasing the number of patients in the study would increase the circuit depth significantly, which comes with several disadvantages including increasing the parameter sizes, and hence the key size, and ciphertexts size, as well as the computation time.

2.2 Secure multiparty computation approach

Kamm et al. [25] propose a solution to address the privacy challenges in genome-wide association studies. Their application scenarios, much like ours, focus on large data collections from several biobanks, and their solutions are based on the same fundamental techniques as ours. However, the setting of Kamm et al. [25] requires all raw genotype, phenotype, and clinical data to be entered to the secure shared database. To the contrary, our setting assumes that only the aggregate values, necessary to identify the significance of a gene-disease relationship (i.e., the contingency tables recording the counts of genotypes vs. phenotypes), are contributed by each biobank. This is a simpler, and more realistic setting, which not only is likely to be implemented in the near future, but also alleviates the computational cost of the proposed solutions. Unlike the approach of Kamm et al. [25], and the alternatives that they suggest, our solution achieves active security with dishonest majority (contrary to the semi-honest security suggested). This means that our protocols tolerate dishonest behavior by the majority of the computing parties, while preserving privacy, and still guarantee the correctness of accepted results. Kamm et al.'s protocols assume that the computing parties –the biobanks– cannot be corrupted, which we consider to be a strong assumption.

Independent and concurrent work by Cho et al. [10] tries to address the same problem as we do in our work, using multiparty computation techniques. They focus on a method that enables the identification and correction for population biases before computing the statistics. However, just like the work of Kamm et al. [25], they make the strong assumption of semi-honest security. In practice, the semi-honest security is not a sufficient security guarantee for GWAS, as attackers who have obtained access to the systems are likely to employ active measures to obtain the data.

Constable et al. [11] present a garbled-circuit based MPC approach to perform GWAS. Their solution can compute in a privacy-preserving manner the minor allele frequency (MAF), and the χ^2 statistic. Similarly to the work of Kamm et al. [25], the framework of Constable et al. [11] requires the raw genotype,

and phenotype data, increasing the workload of the proposed privacy-preserving system. In contrast to our solution, which can scale to hundreds of medical centers contributing data to the GWAS, the solution of Constable et al. [11] only works for two medical centers. Despite the strong security guarantees that our approach offers, which generally presents itself as a tradeoff to efficiency, our proposal is faster than that of Constable et al. [11]. This is also due to the fact that we have optimized the computations of the χ^2 statistic, in such a way that the expensive computations in the privacy-preserving domain, are avoided to the maximum extent possible.

Zhang et al. [42] propose a secret-sharing based MPC approach to solve the same GWAS problem as Constable et al. [11]. Although Zhang et al.'s solution can scale to more than two medical centers contributing data to the GWAS, the approach has the same inherent limitations (e.g., requiring raw genomic data as input) that their application scenario incurs. The works of Zhang et al. [42], Constable et al. [11], and Cho et al. [10] have not considered protecting the aggregate statistic result of the private computation, which –as Homer et al. [23] showed– can be used to breach an individual's privacy. We additionally protect the aggregate statistic result, while at the same time allowing for a public list to be created, showing which SNVs are significant for a certain disease.

3 Distributed GWAS Scenario

In this paper we aim at identifying which mutations are linked to which phenotypes, without compromising the privacy of the patients. Specifically, there are K centers (hospitals) who each have genotype (SNV), and phenotype (trait) data. For a single genotype-phenotype pair a center k has a 2×2 contingency table[6] of the counts of patients for all 4 possible combinations of genotype, and phenotype (see Table 1). The goal is to perform a privacy-preserving computation that adds together all contingency tables from individual centers, then computes the Pearson's χ^2 test statistic [35], and finally reveals a boolean value indicating whether the computed statistic is larger than a predetermined significance threshold t. This threshold is chosen based on the p-value, and the correction for multiple hypothesis testing. For example, using significance level 0.01 with Bonferroni correction for 10 million tests results in $t = 37.3$, and for 100 million tests $t = 41.8$.

We propose two different methods for carrying out the χ^2 test without disclosing the input, and intermediate values. The first method performs all computations on *homomorphically* encrypted data, while the second applies techniques of *secure multiparty computation* to achieve the same goal. Both methods follow

[6] Our method can be also extended to contingency tables of larger size.

the same general outline, presented below. The first step is to encrypt (or secret share) all the input tables from the centers, and securely compute the aggregate contingency table

$$O_{ij} = \sum_{k=1}^{K} O_{ij}^{(k)}, \tag{1}$$

where $O_{ij}^{(k)}$ is the data from k-th center. This step is straightforward in both methods.

Next to determine the significance of the relation between a mutation, and a phenotype, we calculate the Pearson's χ^2 test statistic [35] on the aggregated contingency table O, and check whether this value is above the threshold t. The Pearson's χ^2 statistic is given by the following formula:

$$\chi^2 = \sum_{i,j \in \{1,2\}} \frac{(O_{ij} - model_{ij})^2}{model_{ij}}, \tag{2}$$

where $model_{ij} = (RT_i \cdot CT_j)/N$ with $RT_i = O_{i,1} + O_{i,2}$ being the row total, $CT_j = O_{1,j} + O_{2,j}$ being the column total, and N the total number of patients.

	phenotype	$\neg$*phenotype*	
genotype	$O_{1,1}$	$O_{1,2}$	$RT_1 = O_{1,1} + O_{1,2}$
$\neg$*genotype*	$O_{2,1}$	$O_{2,2}$	$RT_2 = O_{2,1} + O_{2,2}$
	$CT_1 = O_{1,1} + O_{2,1}$	$CT_2 = O_{1,2} + O_{2,2}$	$N = CT_1 + CT_2 = RT_1 + RT_2$

Table 1: Representation of a contingency table containing the number of observed genotypes i per phenotype j noted by $O_{i,j}$. In the table we also calculate the Row Totals (RT_i), Column Totals (CT_j), as well as the grand total (N).

Since division is a costly operation in both the homomorphic domain, and secret shared domain, we will rewrite the formula of the χ^2 statistic as follows:

$$\begin{aligned}\chi^2 = &\frac{RT_1 \cdot CT_1 \cdot (N \cdot O_{2,2} - RT_2 \cdot CT_2)^2}{N \cdot RT_1 \cdot RT_2 \cdot CT_1 \cdot CT_2} + \frac{RT_1 \cdot CT_2 \cdot (N \cdot O_{2,1} - RT_2 \cdot CT_1)^2}{N \cdot RT_1 \cdot RT_2 \cdot CT_1 \cdot CT_2} \\ &+ \frac{RT_2 \cdot CT_1 \cdot (N \cdot O_{1,2} - RT_1 \cdot CT_2)^2}{N \cdot RT_1 \cdot RT_2 \cdot CT_1 \cdot CT_2} + \frac{RT_2 \cdot CT_2 \cdot (N \cdot O_{1,1} - RT_1 \cdot CT_1)^2}{N \cdot RT_1 \cdot RT_2 \cdot CT_1 \cdot CT_2}.\end{aligned} \tag{3}$$

As a final step, we need to compare whether $\chi^2 \geq t$. To do that, we calculate the numerator, and denominator of the fraction in Equation (3), separately. Subsequently, we multiply the denominator of the fraction with the threshold value t, and finally check inequality (4), without revealing any of the private inputs in the contingency tables.

$$\begin{aligned} RT_1 \cdot CT_1 \cdot (N \cdot O_{2,2} - RT_2 \cdot CT_2)^2 + RT_1 \cdot CT_2 \cdot (N \cdot O_{2,1} - RT_2 \cdot CT_1)^2 \\ + RT_2 \cdot CT_1 \cdot (N \cdot O_{1,2} - RT_1 \cdot CT_2)^2 + RT_2 \cdot CT_2 \cdot (N \cdot O_{1,1} - RT_1 \cdot CT_1)^2 \\ \stackrel{?}{\geq} t \cdot (N \cdot RT_1 \cdot RT_2 \cdot CT_1 \cdot CT_2). \end{aligned} \tag{4}$$

This computation is repeated for every phenotype-genotype pair, and the results are aggregated in a public table indicating whether a mutation is significant for a particular phenotype, or not. Since the price of DNA sequencing has decreased a lot, we assume new data will keep becoming available. Taking this new data into account for the computation of our public table, requires running our protocols anew, and it will change the table results. Therefore, we propose to make the table dynamic. There will be a fixed time interval, which allows the centers to gather more data and include this data in their contingency tables. The new table values will then be encrypted/secret shared and the computation of the fresh public table will be executed, after which the new results will be published.

3.1 Efficient Masking-Based Comparison

To the best of our knowledge the state-of-the-art techniques to perform secure comparisons, both in the homomorphic, and in the secret shared domain, require bitwise operations on the secret inputs, which have a high total cost. To allow for a practically efficient implementation of our solution, we consider a masking technique to perform the comparison instead of the bit-decomposition of our inputs. By masking the values we need to compare, we can later securely reveal the masked result upon decryption, since the mask will hide the original secret value. Hence, masking allows us to perform the comparison without revealing the values we want to keep secret. Comparing two values x and y can be done by comparing their difference with zero. Our mask for the value $x-y$ consists of multiplying this value with a positive random number. We require the multiplier to be positive to preserve the original relation of our difference $x - y$ with zero. The second step is adding another random number (different than the previous one) to the already multiplied result. We require this random number to be smaller than the first one, again to preserve the original relation with zero. Let us denote the masked difference with $\widehat{x-y}$, then for two positive random numbers r and r', with r' in the range $[1, r)$, our proposed masking is given by $\widehat{x-y} = r \cdot (x - y) + r'$. In our setup we are working with homomorphic (or secret shared) values, so this masking has to be performed on encrypted (or secret shared) values. For an integer x, we denote $[\![x]\!]$ either the homomorphic

encryption of x or its secret shared value. Masking in the homomorphic or secret shared domain will then be computed as $[\![\widehat{x-y}]\!] = [\![r]\!] \cdot [\![x-y]\!] + [\![r']\!]$, with r and r' random numbers satisfying the following condition: r is selected to be a positive integer number (bounded properly so as to fit the largest possible input sizes our framework can handle), and then r' is randomly selected in the range $[1, r)$ (i.e., such that $r' < r$) . Afterwards the masked value is revealed by respectively decrypting or opening the calculated value. Depending on the sign of $(\widehat{x-y})$ we can deduce the relationship between x and y (i.e., if $(\widehat{x-y}) > 0$ then $x > y$, otherwise $x < y$).

Given properly selected r, and r', the correctness of this masking-based comparison is straightforward. To ensure preservation of the security and correctness of the masking, we require one of the medical centers to properly select r, and r' within suitable bounds. Note that this requirement does not increase the level of trust we need to put in the medical centers (nor does it reduce the security of the system). We already trust the medical centers to provide our privacy-preserving system with their correct inputs. Upon selection of the values r, and r', the medical center in question homomorphically encrypts these values, or secret shares them to the computation servers, along with its own contributed inputs.

The proposed type of masking, which allows us to perform the comparison, could leak information about the secret input to the inequality, which in our case is the difference $x - y$. Since none of the mutliplicative, or additive blindings we deploy are performed based on a uniformly sampled randomness in the space where the secret value lives, perfect security cannot be guaranteed. We also did not account for a statistically larger space from which to sample either of the multiplicative or additive randomness, hence the protocol including the masked comparison is not statistically simulatable. Despite this information leakage, we consider the combination of multiplicative and additive blinding to offer sufficient privacy for this particular application scenario, where only information about the *range* of the χ^2 statistic may be leaked, and we reiterate that a lower bound on this value (namely the threshold we compare against) is already publicly revealed by the protocol.

The information leakage issue becomes more crucial, when the system is queried multiple times with the same input. However, in our scenario, the proposed system cannot be queried at will. We suggest the calculation of a table listing all possible phenotypes, and all possible mutation positions, which will become public. This table will be computed on updated input data after a fixed time interval. While constructing the table we select r and r' at random for each contingency table, thus the random values r and r' will only be used once with certain input values. After the fixed time interval, we expect the input values to be changed, so we repeat the whole setup and select new random values for

each contingency table. Hence, by recomputing the table at fixed times and not allowing users to query, we ensure that no information is leaked by our system.

Let us for completeness briefly discuss the leakage that occurs if a party observing the masked result of the inequality check can submit multiple queries on the same inputs and obtain the masked values for these queries. If this would be possible, an adversary would be able to approximate the value of $x - y$ from the obtained list of masked values. The maximum of this observed list will be close to the bound set for the randomness r times the difference $x - y$. Hence, by deduction, if we divide the maximum observed masked value by the upper bound on r, we will get a good approximation of the value $x - y$.

In the event of a malicious party being able to observe the intermediate values revealed by our approach (i.e., the value of the masked difference), and given that this malicious party can trigger multiple computations of the same table entry, one can prevent the aforementioned leakage by selecting the random values r and r' once per table entry, and keep them thereafter fixed, until the actual inputs to the protocol (contributed by each medical center) change.

4 Homomorphic Encryption Approach

4.1 Setup and security assumptions

To solve the problem described in Section 3 with homomorphic encryption, we need multiple parties, as indicated in Figure 1. The steps of the process depicted in Figure 1 are as follows. In the first step, the decryptor will select the secret key, and associated public key for the homomorphic encryption, and make the public key available to all medical centers. Then, all the medical centers will encrypt their contingency tables with the given public key, and send these encryptions to the computation server. Upon receiving all contingency tables, the computation server will first add them to construct the aggregated contingency table, and subsequently perform the operations of the Pearson χ^2 test. Then, the computation server will send the result, which is masked with the technique described in Section 3.1 to the decryptor, who uses the secret key to decrypt the masked value, and performs the comparison.

It is important to note that in this model we trust the decryptor to decrypt the masked values, and post the corresponding correct yes/no value into the public table. Since the decryptor only decrypts masked values, the decryptor can only deduce the yes/no answer which will become public, anyway. No other information about the χ^2 value is revealed to the decryptor. If the system would be adapted to reveal the actual χ^2 value, the party receiving the encrypted result would first have to authenticate itself to make sure that it is a trusted entity (like

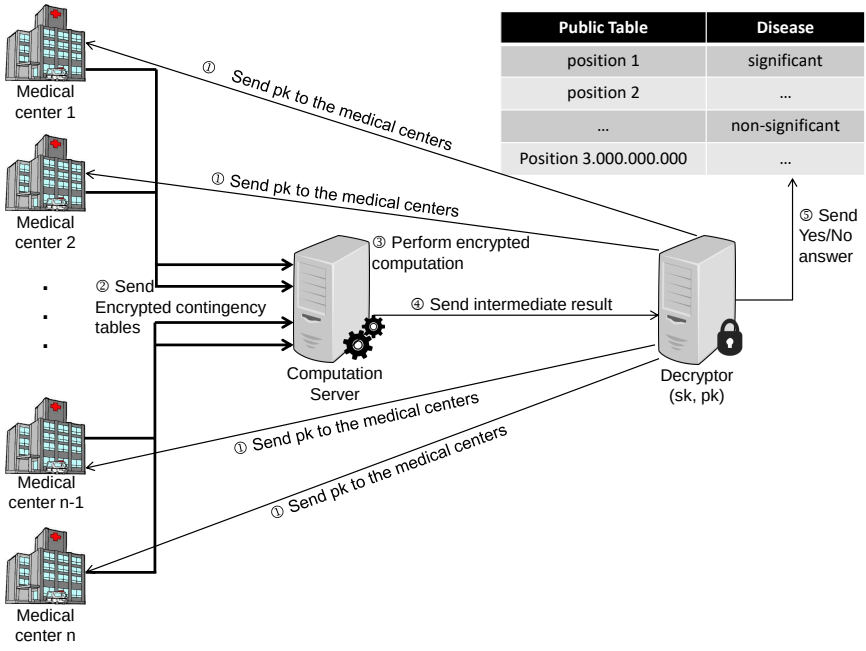


Fig. 1: A schematic representation of the homomorphic scenario.

Before the execution of the protocol, the decryptor generates a valid public and secret key pair for the homomorphic encryption scheme. Step (1) of the protocol is to send the generated public key pk to all participating medical centers. Then, the medical centers compute their local contingency tables, encrypt them with the received public key, and send them to computation server in step (2). Step (3) is the actual secure computation of the encrypted, and masked χ^2 value, which is then sent to the decryptor in step (4). By decrypting the masked χ^2 value (using the secret key sk), the decryptor can only determine whether the result is significant or not, which is published in a public table in step (5).

a medical doctor, for example). If this authentication is considered insufficient by the medical centers contributing their data, they could still prevent the authenticated party from being a single point of trust by introducing a multiparty computation to perform the decryption based on a secret shared decryption key. The solution based on homomorphic encryption does rely on the following two security assumptions:

- The computation server is honest but curious: It will follow the stated protocol to provide the desired functionality, and will not deviate, nor fail to return the results. The computation server can however monitor the result of every operation it performs. This assumption is reasonable for an economically motivated cloud service provider. The cloud is motivated to provide excellent service, yet it would take advantage of extra available information.

- We only need the decryptor to perform the comparison. He should not be allowed to see the input values, since he has the key to decrypt them. Therefore we presume that the communication between the centers, and the computation server is hidden from the decryptor. This can be achieved by performing the communication over authenticated, secure channels. An alternative way to solve this is by introducing the multiparty computation for the decryptor. Each party only has a part of the decryption key, and hence will never be able to decrypt the values of the encrypted contingency table.

For the homomorphic evaluation of the χ^2 statistic we use the FV scheme, introduced by Fan and Vercauteren [21]. Moreover, we base our implementation on the FV-NFLlib software library [13] in which the FV homomorphic encryption scheme is implemented using the NFLlib software library developed for performing polynomial arithmetic computations (as described in [1], and released in [14]).

4.2 Preliminaries

The Fan-Vercauteren SHE scheme The Fan-Vercauteren SHE scheme is a scale invariant SHE scheme whose hardness is based on the ring learning with error problem (RLWE) [32]. It operates on polynomials of the ring $R = \mathbb{Z}[X]/(f(X))$ with $f(X) = X^d + 1$ for $d = 2^n$.

The FV scheme makes use of a plaintext space R_t with R the polynomial ring defined above, and $t > 1$ a small integer modulus. Each coefficient of a plaintext polynomial is computed modulo t. The ciphertext space consists of a pair of elements of R_q with R the polynomial ring defined above, and $q > 1$, an integer modulus much larger than the plaintext modulus t. A homomorphic

encryption scheme consists of a standard encryption scheme specifically constructed to enable additions, and multiplications in the ciphertext domain. The key generation, and encryption algorithms require elements sampled from two probability distributions defined on R. The secret key of our scheme is sampled from χ_{key}, and for encryption some error polynomials are sampled from an error distribution χ_{err}. These probability distributions in combination with the degree d of the defining polynomial f of R, and the size of the integer q determine the security of the FV scheme.

Given the parameters d, q, and t, and distributions χ_{key}, and χ_{err}, and using bold notation for a vector of two polynomials, we define the encryption, and decryption mechanism of the homomorphic encrytion scheme introduced by Fan and Vercauteren:

- Encrypt($\mathbf{pk}$,m): By multiplying the message $m \in R_t$ with $\Delta = \lfloor q/t \rfloor$ we transfer the message m to the ring R_q. To hide the message we sample the error polynomials $e_1, e_2 \in \chi_{err}$, and $u \in \chi_{key}$, and compute the polynomials $c_0 = \Delta \cdot m + bu + e_1$, and $c_1 = au + e_2$. Both the polynomials c_0, and c_1 belong to R_q, and together they form the ciphertext $\mathbf{c} = (c_0, c_1)$ of the FV scheme.

- Decrypt(sk, $\mathbf{c}$): First compute $\tilde{m} = [c_0 + s \cdot c_1]_q$, then by scaling down the coefficients of $\tilde{m}$ by Δ, and rounding the results we recover the message m.

Given a ciphertext $c = (c_1, c_2)$ we can write the $\tilde{m}$ of the decryption algorithm as $[c_0 + c_1 s]_q = \Delta \cdot m + e$, with e the noise in the ciphertext. From this equation one can clearly see that if the noise e grows too large, the decryption algorithm will fail to output the original message m correctly. The property of ensuring that decryption results in the original message is called the correctness of the encryption scheme.

Every homomorphic operation will cause the noise in the ciphertexts to increase. Knowing the computations we want to perform in advance enables us to optimize the order of the computations for the sake of minimizing the noise growth. In addition it enables us to make an estimation of the noise present in the result, and hence allows us to determine parameters that can deal with this noise.

Preprocessing of the data to improve the performance of the homomorphic computations The plaintext space of the FV scheme consists of the polynomial ring R_t. Hence the first thing that needs to be done is encode the given integer data values into polynomials in the ring R_t. We achieve this by applying the w-NIBNAF encoding introduced in [6]. The main idea of w-NIBNAF encoding goes back to Dowlin et al. [18] (see also [19, 30, 34]), and was analyzed

in more detail by Costache et al. [12]. It consists of expanding the given number θ with respect to a base b_w, and replacing this base with the symbol X. So the encoding of a number θ is given by:

$$\theta = a_r X^r + a_{r-1} X^{r-1} + \cdots + a_1 X + a_0 - a_{-1} X^{d-1} - a_{-2} X^{d-2} - \cdots - a_{-s} X^{d-s} \ .$$

For our encoding we use a well chosen non-integral base such that the encoding of an input value results in a polynomial with coefficients in the set $\{-1, 0, 1\}$, with the property that each set of w consecutive coefficients has no more than one non-zero coefficient. Hence encoding our inputs with w-NIBNAF leads to sparse polynomials.

Starting our computations with sparse polynomials with coefficients in the set $\{-1, 0, 1\}$ leads to small coefficients in the resulting polynomial. Therefore it enables us to work with a smaller plaintext modulus t, which improves the performance of the homomorphic encryption scheme. Bigger values for w leads to longer, and sparser encodings, which reduce the minimum size for t even further. However, we have to ensure that when all the computations are done, decoding will still provide the correct answer. Therefore, we need to carfully select our base b_w, which also determines the value of w.

Selecting the optimal parameters The three main concepts that will affect the selection of our parameters are:

- the security of the somewhat homomorphic FV scheme
- the correctness of the somewhat homomorphic FV scheme
- the correctness of the w-NIBNAF encoding

To take the security restrictions into account, we rely on the work by Albrecht, Player, and Scott [3], and the open source LWE hardness estimator implemented by Albrecht [2]. The latter estimates the hardness of the LWE problem based on three given parameters: the dimension d, the ciphertext modulus q, and a parameter α, which is related to the error distribution χ_{err}. It takes into account the currently known attacks on the learning with error problem.
The parameters that satisfy the restriction of the security implications of a security level of 90 bits are $q = 2^{186}$, $d = 4096$, $\sigma = 265$[7]. From the description of the FV scheme in Section 4.2, we know that for a ciphertext to be decrypted correctly the error cannot grow too much. Therefore we make an estimation of the infinity norm of the error in the ciphertext resulting from computing our

[7] σ determines the error distribution χ_{err}.

circuit, and select parameters that keep the error small enough. This results in an upper bound for t. Since this upper bound is constructed specifically for our circuit it will depend amongst other things on the number of centers that deliver data to the computation server.

For correctness of the w-NIBNAF decoding, it is important to estimate the size of the coefficients of the encodings after performing the necessary operations. This leads to a lower bound on the plaintext modulus t. The security, and correctness of the FV scheme on one hand, and the correctness of the w-NIBNAF encoding on the other hand, set conflicting requirements for t. In order to solve this conflict, we make use of the Chinese Remainder Theorem to decompose the plaintext space, following the idea of [7, §5.5]. This implies that we choose t to be the product of small prime numbers $t_1,\ t_2,\ \ldots,\ t_n$, with $\forall i \in \{1,\ \ldots,\ n\} : t_i \leq t_{\max}$, and $t = \prod_{i=1}^{n}\ t_i \geq t_{\min}$, where $t_{\max}$ is determined by the security, and correctness of the FV scheme, and $t_{\min}$ by the correctness of the $w-$NIBNAF decoding.

4.3 Privacy-Preserving Homomorphic Chi-Squared Thresholding Algorithm and Parameters

In this section, we combine all information given before in order to construct the algorithm needed to perform the privacy-preserving χ^2 thresholding in the homomorphic setting. Algorithm 1 lists the computations for the algorithm corresponding to the order in which they need to be performed, and mentions the parties that need to perform them. Hence the steps from Figure 1 are described in more detail in Algorithm 1, and the combination of these two gives a clear picture of our homomorphic solution for the privacy-preserving χ^2 thresholding.

For the homomorphic solution we consider two scenarios: (1) the case in which we compute the numerator and denominator of the χ^2 statistic seperately, and decrypt them both, and (2) the case where we use the masking technique, and decode the masked value to determine the "True/False" answer to the significance question. As mentioned before, there are many parameters we need to set, and their values depend highly on the homomorphic circuit we perform and the values of other parameters. We selected parameters for different scenarios in which we fix the number of patients per center to 10000 and vary the number of centers from 20 to 100. The parameters are listed in Table 2 and 3. We use these paramters in our implementation later to assess the performance by measuring the computation time, and communication cost for each of the three parties mentioned in Algorithm 1.

The value of w and the splitting degree are dependent on the size of the numbers we need to encode, but independent of the number of centers included

Algorithm 1 Privacy-preserving Homomorphic Chi-squaredTest

Medical center:

input: $O[2][2]$: the observed values of the 2×2 contingency table (mutation vs. disease)

for C_k medical center k **do**

1. Encode each value $O^k_{i,j}$ for $i = 1, 2$; $j = 1, 2$ using the w-NIBNAF encoding technique .
2. Transform this encoding to the plaintext ring R_{t_i} for all i such that $t = \prod_i t_i$.
3. Encrypt the plaintext polynomials using the Fan-Vercauteren SHE scheme to obtain ciphertexts $\mathbf{c^k_{i,j}}$.

end for

Computation center:

input: encrypted contigency table values $\mathbf{c^k_{i,j}}$, for $k = 1, ..., Nc$, with Nc the number of centers contributing data

4. Compute the four values of the aggregated contingency table $\mathbf{c_{m,n}} = \sum_{k=1}^{Nc} c^k_{m,n}$.
5. Compute the χ^2 numerator α and denominator β homomorphically
6. Compute the difference of the χ^2 numerator and the product of the encrypted χ^2 threshold T and denominator, and mask the computed difference with the random values r and r': $MR = r \cdot (\alpha - T \cdot \beta) + r'$

Decryptor:

input: MR the encrypted masked output value of the homomorphic circuit

7. Decrypt the masked result MR
8. Use the inverse CRT ring isomorphism to transfer the plaintext polynomials to the ring R_t.
9. Decode the w-NIBNAF polynomial, and evaluate the result in the correct basis b_w to get the value $MaskedDifference$ of the masked result.
10. Determine the significance based on the sign of $MaskedDifference$:

if $MaskedDifference > 0$ **then**

return 1

else

return 0

end if

in the calculation. Therefore the value of w and the splitting degree will be the same for all scenarios. Without comparison, they are respectively $w = 271$, and splitting degree = 3166. The parameters that differ for different number of centers in scenario (1) is the value for t, and its CRT factors. These are listed in Table 2. For scenario (2) in which we perform the masked comparison, the value of w will be different for each different number of centers participating in the computations, because we have to encode the random variables r and r', which have sizes depending on the number of centers included in the computations. The parameters for the second scenario are listed in Table 3.

Table 2: Parameter selection for scenario (1)

Centers	**Patients**	**t**
20	200000	$17431 \cdot 17443 \cdot 17449$
40	400000	$1718713 \cdot 1718719 \cdot 1718723$
60	600000	$1558103 \cdot 1558129 \cdot 1558177$
80	800000	$1453043 \cdot 1453057 \cdot 1453061$
100	1000000	$1376213 \cdot 1376231 \cdot 1376237$

Table 3: Parameter selection for scenario (2)

Centers	**Patients**	**bit-length random value**	**w**	**splitting degree**	**t**
20	200000	100	118	3625	$12253 \cdot 12263 \cdot 12269$
40	400000	106	113	3629	$885127 \cdot 885133 \cdot 885161$
60	600000	110	111	3644	$802651 \cdot 802661 \cdot 802667$
80	800000	112	110	3653	$747811 \cdot 747827 \cdot 747829$
100	1000000	114	108	3651	$707801 \cdot 70813 \cdot 707827$

4.4 Implementation and performance analysis

In order to assess the practical performance, and verify the correctness of the selected parameters of the homomorphic scenario, we implemented the privacy-preserving χ^2 computation using the FV-NFLlib software library [13]. Our presented timings are obtained by running the implementation on a computer equipped with an Intel Core i5-4590 CPU, runnning at 3.30 GHz. We executed the program 10 times per case, and calculated the average execution time for our timing results. To evaluate the scalability of our protocol we have considered the cases where our system receives data from 20, 40, 60, 80, and 100 medical centers, respectively. We assume each medical center to contribute data of 10000

subjects (i.e., the total number of subjects per case is 200000, 400000, 600000, 800000, and 1000000, respectively). In order to measure these timings we used the parameter set corresponding to the same scenario, so Table 2 for scenario (1) and Table 3 for scenario (2).

For scenario (1) we compute the numerator and denominator seperately, and do not perform the masked comparison with the threshold value. The CPU time needed for the hospitals to encrypt the four values of the contingency table is the same for any number of centers considered in the experiment. For our selected parameters this is 15.2 ms. Also the time to decrypt the numerator and denominator of the χ^2 value is the same for any number of centers contributing to the experiment. The average time we measured during our experiments is 38.8 ms. The timings of the computation server in scenario (1) are the times needed to perform the calculations for computing the numerator and denominator of the χ^2 statistic. These timings are dependent on the number of centers that participate in the computation. Therefore we list them in Table 4. We see that the timings for increasing centers do not differ significantly. This is consitent with the fact that homomorphic additions are not the most time consuming part of our computations.

For scenario (2) the encryption time does not depend on the number of centers either, since the centers can perform the encryption in parallel. The measured encryption time for one contingency table in scenario (2) is 17.1 ms. The time to decrypt the result does not depend on the number of centers participating in the computation either. However it is smaller than for scenario (1), since now we only have to decrypt one value instead of two. The measured decryption time for scenario (2) is 21.1 ms. The timings for the computation server are listed in Table 4, since these timings are dependent of the number of medical centers participating. Here we see the same trend as for scenario (1): the timings do not increase linearly in the number of medical centers. Interestingly, in this scenario, the number of patients contributing data does not affect the efficiency of our protocol, which further supports the important aspect of scalability in GWAS. As long as the total frequencies in the contigency tables do not grow larger than the HE parameters allow for correctness, the efficiency of the protocol remains intact from an increasing number of patients.

From Table 4 one sees that the timings for the computation server do not differ much between both scenarios. This is because addition and multiplication with a constant (which are the extra operations we need to compute the masked value) are not the most time consuming homomorphic operations. Hence our masked comparison gives an efficient solution for keeping the χ^2 value private. We can also conclude that considering CPU time, our solution scales really well for increasing number of medical centers participating in the computation.

Table 4: CPU time of the computation server for the homomorphic solution using 1 CPU core.

Centers	Patients	scenario (1)	scenario (2)
20	200000	1.40 s	1.48 s
40	400000	1.48 s	1.52 s
60	600000	1.44 s	1.53 s
80	800000	1.47 s	1.56 s
100	1000000	1.49 s	1.56 s

For the homomorphic setup, there is no communication cost during the computations. The communication cost comes from sending values from each of the three parties to the next. We have three points of communication: the public key has to be sent from the decryptor to the medical centers; the encrypted values of the contingency tables have to be sent from the medical centers to the computation server; and the result has to be sent from the computation server to the decryptor. The communication cost is similar for both scenarios since for both scenarios the size of one ciphertext will be the same, and we only need to sent ciphertexts form one party to another. The size of the public key that needs to be sent to the different medical centers is 186 kB. The data needed to send one contingency table to the computation server is 2.1 MB. The communication cost between the medical centers, and the computation server is the number of centers participating times the number of data needed to send one contingency table. So this communication cost increases linearly in the number of centers contributing to the computation. In scenario (1) we send both the numerator as the denominator from the computation server to the decryptor, which results in a communication cost of 1.8 MB. In scenario (2) we only have to send one value, which gives a communication cost of 0.54 MB.

5 Secure Multiparty Computation Approach

To address the challenge of disease gene identification using secure multiparty computation techniques, in the setting described in Section 3, we deploy MASCOT [27]. We selected MASCOT [27] as the most suitable multiparty computation solution, because it is currently the most efficient proposal, offering malicious static security with a dishonest majority. This means that any number of the computing parties may deviate from the protocol execution, and this will be detected without leaking information, other than what the correct protocol execution would reveal. Corruption may only occur prior to the beginning of the protocol execution, affecting up to $n - 1$ (out of the n) computing parties.

5.1 Setup and Security Assumptions

For our multiparty computation approach, we first need to determine the number of computation servers n ($n \geq 2$) that we have at our disposal. Given that the underlying protocol offers security against any coalition of $n - 1$ computation servers, we consider the security of the whole system to increase as the number of computation servers increases. However, the number of computation servers is inversly proportional to the efficiency of the solution. Therefore, we consider that three computation servers is an adequate number of servers, both from an efficiency/plausibility perspective, and from a security perspective. If any two of the three computation servers that we assume get compromised, or otherwise behave dishonestly, or even collude, the solution still guarantees input privacy, and does not accept incorrect results.

We assume a preprocessing phase that can take place offline, at any moment prior to the actual protocol execution. This is to create the necessary randomness for the medical centers to contribute their inputs in a secret shared manner to the computation servers. In addition, the preprocessing phase creates authenticated randomness to be used in the online phase, so as to boost the efficiency of computing multiplications on the shares, which requires interaction amongst the servers.

The medical centers that wish to contribute their private inputs, first need to agree on a common format for this data (e.g., what is the order of sending the contingency tables). Then, they need to secret share their contingency tables to the three computation servers, which can also be pushed to an offline, preprocessing phase. Given that all contributing medical centers have shared their private contingency tables to the computation servers, the online phase starts. During the online phase the servers perform both local, and interactive secure computations, and they finally reveal per contingency table whether the relationship between a mutation at a certain DNA position, and a disease is significant or not, without disclosing further information on the underlying data. A schematic representation of this approach is presented in Figure 2.

5.2 Preliminaries

Additive Secret Sharing A secret sharing scheme is a protocol, which allows (some of) the protocol participants to share their secret inputs amongst all other protocol participants, in such a way that nothing is revealed to the individual participants about the secret input. In some instances of secret sharing schemes, a *subset* of the protocol participants, called the qualified set, can reconstruct the original secret input, when they engage in the reconstruction protocol. Other

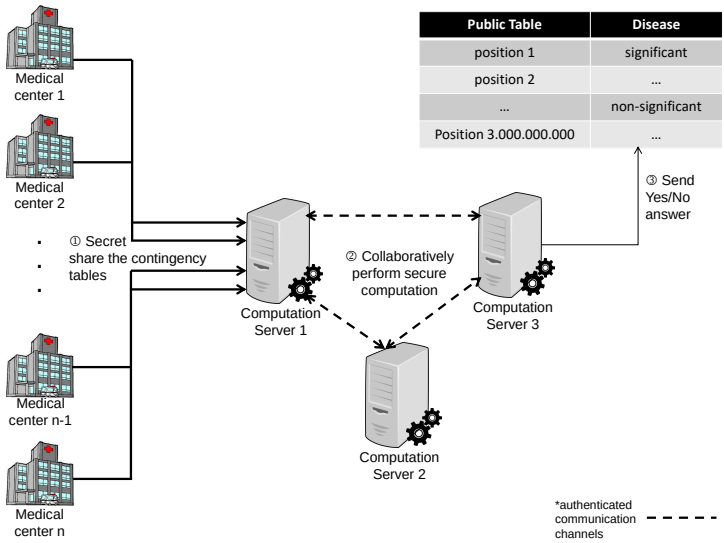


Fig. 2: A schematic representation of the multiparty computation scenario
Any time prior to the protocol execution each of the medical centers computes their local contingency tables, and secret shares them to the three computation servers, as indicated in step ① of the protocol. In step ②, the computation servers securely compute the χ^2 value, and perform a secure comparison to determine whether the value is significant or not. This reveals no information about the inputs, or the actual χ^2 value to the individual computation servers. In step ③ the computation servers reconstruct the final result, which indicates significance or non-significance, by combining their individual secret shares, and they publish this result in the public table.

schemes, such as additive secret sharing, require *all* protocol participants to contribute their shares for the reconstruction protocol to work.

Additive secret sharing is essentially masking the input of a protocol participant x by subtracting a random value r from it. Given that the value of r is only known to the inputting participant, and that the rest of the participants hold shares $[\![r]\!]$ of this value (which can be done in a preprocessing stage), secure shares of x can be created as follows: The inputting party computes $\epsilon = x - r$, and broadcasts it to the rest of the participants. All other parties can now compute their own shares of the input x as $[\![x]\!] = [\![r]\!] + \epsilon$. It is easy to see that this scheme enjoys an additively homomorphic property, allowing additions of shares, and linear functions to be directly computed locally on the shares. Hence, no communication amongst the protocol participants is required to perform additions on these additively secret shared values.

Oblivious Transfer Oblivious Transfer (OT) is a cryptographic primitive, which allows a sender to transfer only one (or none) out of many values to a receiver, while remaining oblivious as to which value has been received (if any). Oblivious transfer was introduced by Rabin [36] in 1981. Basic OT constructions make use of public key encryption primitives to allow the aforementioned functionality. More precisely, an 1-out-of-2 OT requires the sender to be in possession of a public/private key pair, generate, and send two random messages to the receiver, decrypt two messages, and send two messages to the receiver. The receiver obtains the public key of the sender, along with the two random messages that the latter has selected, performs one encryption, and one blinding, sends the resulting message to the sender, and finally inverts the blinding to retrieve the desired message. For more information on basic OT, we refer the reader to the work of Even et al. [20].

Many works have considered extending OT so as to make it practically efficient, with the most notable work of Ishai et al. [24]. Under the assumption that a random oracle H, which can be instantiated by a hash function family, exists, Ishai et al. [24] show how to perform only a few OTs from scratch, and then be able to perform many additional OTs at the cost of a constant number of invocations of the random oracle. The actively secure version of their protocol comes with an increase at the cost of the protocol by a factor σ, for σ a statistical security parameter. More recently, Keller et al. [26] presented an actively secure OT extension protocol, where malicious security comes at negligible extra cost.

Correlated Oblivious Product Evaluation - COPE The Correlated Oblivious Product Evaluation (COPE) protocol presented by Keller et al. [27] is essentially a generalization of Ishai et al.'s protocol [24] to the arithmetic case

(instead of the original binary). The protocol is executed between two parties, and allows them to obtain an additive sharing of the product $x \cdot \Delta$, where the sender holds $x \in \mathbb{F}$, and the receiver holds $\Delta \in \mathbb{F}$. COPE is based on Gilboa's oblivious product evaluation [22], where the parties run k sets of 1-out-of-2 OTs, on k-bit inputs. The proposed product evaluation is correlated in the sense that the one party's input Δ is fixed at the beginning of the protocol for many protocol runs. After a one-time expensive initialization of the COPE protocol, the extension step, generating fresh OTs without the public-key crypto extensive costs, can be repeated several times on new inputs x.

MASCOT Online Phase The online phase of MASCOT [27] is essentially the same as the one of the SPDZ protocol [17, 16]. This family of protocols uses additive secret sharing, allowing additions, and linear functions to be computed locally by the protocol participants, without requiring communication. To achieve active security, information-theoretic MACs are being used, which provide authenticity, and integrity of the messages. A secret shared value x, shared amongst n parties, is represented as follows:

$$[\![x]\!] = (x^{(1)}, ..., x^{(n)}, m^{(1)}, ..., m^{(n)}, \Delta^{(1)}, ..., \Delta^{(n)}), \tag{5}$$

where, $x^{(i)}$ is the random share, $m^{(i)}$ is the random MAC share, and $\Delta^{(i)}$ is the MAC key share, such that $m = x \cdot \Delta$.

To perform multiplications of secret shared values, multiplication triples à la Beaver [5] from the preprocessing phase are required, with which the parties can compute shares of the required product. The multiplication triples are of the form: $([\![a]\!], [\![b]\!], [\![c]\!])$, with a, b uniformly random, and $c = a \cdot b$. More precisely, to compute $[\![z]\!] = [\![x \cdot y]\!]$, on input $[\![x]\!]$, $[\![y]\!]$, and given a multiplication triple $([\![a]\!], [\![b]\!], [\![c]\!])$, the parties compute $\epsilon = [\![x]\!] - [\![a]\!]$, and $\rho = [\![y]\!] - [\![b]\!]$, and open these two values. Then, they compute $[\![z]\!] = [\![c]\!] + \epsilon \cdot [\![b]\!] + \rho \cdot [\![a]\!] + \epsilon \cdot \rho$.

MASCOT Offline Phase - Preprocessing The goal of the preprocessing phase is to generate random values for the parties who wish to contribute inputs –so as to allow them to mask their inputs in an authenticated manner–, and multiplication triples –so that multiplication of secret shared values can be efficiently implemented in the online phase–.

The preprocessing is based on Oblivious Transfer techniques, and more precisely the authors of MASCOT [27], have generalized the OT extension idea presented by Ishai et al. [24] to the arithmetic circuit case. For every party that wishes to contribute an input, the COPE protocol is executed, to create an authenticated version of the input, based on the global MAC key Δ. Creation of (authenticated) additive shares is straightforward. Recall that both the shares,

and their MACs are linear. Hence, computing linear functions on (authenticated) shared values is also straightforward. To ensure active security, the party that wishes to contribute an input first authenticates a random input x_0 together with the actual inputs m. Then, the party opens a random linear combination of the inputs including x_0, and all other parties check the MAC on this linear combination. This way the party contributing input is committed to them, and the actual inputs are masked by the random input x_0.

For the triple generation, the parties invoke an OT to compute the secret sharing of $b \in \mathbb{F}$, and $\mathbf{a} \in \mathbb{F}^\tau$, where $\tau \geq 3$ is a security related parameter, meaning that they run τ copies of the basic two-party COPE per pair of parties. This ensures that $\mathbf{a}$ has enough randomness to produce a triple. To protect against malicious behavior, the parties sample two sets of random coefficients $\mathbf{r}$, and $\hat{\mathbf{r}} \in \mathbb{F}^\tau$, from which they generate two triples with the same b component. Upon authentication of their shares, the parties ensure correctness of one of the triples, by sacrificing the other triple.

5.3 Privacy-Preserving Chi-Squared Thresholding Protocol

Our protocol calculates the units of inequality (4) using MASCOT [27]. For the inequality check, we apply the masking technique described in Section 3.1. In addition to the masking-based comparison, we have also implemented the protocol using the standard, bit-decomposition based secure comparison, as implemented in SPDZ-2 [8]. We have also made a non-secure implementation avoiding completely the comparison. All these cases are presented in Section 5.4, analyzing their performance. The online phase of the masking-based version of our protocol is detailed in Algorithm 2.

In our setting, we consider the computing parties that actually execute our protocol (i.e., the computation servers), different from the parties contributing their inputs (i.e., the medical centers), as shown in Figure 2. For the offline phase, together with the preparation of the triples, and randomness discussed in Section 5.2, we wish to perform the required preprocessing that will allow the medical centers to correctly contribute their inputs, without compromising privacy. To do that we use the protocols proposed by Damgård et al. [15]. First we use the Output Delivery protocol to reveal a preprocessed random value r only to the inputting party, who can then broadcast his masked input $x - r$ to the computing parties. Based on this value, and the preprocessed randomness r, the servers can locally compute their share of x, as $[\![x]\!] = (x - r) + [\![r]\!]$.

5.4 Implementation and Performance Analysis

We have built a proof of concept implementation of our MPC approach using the platform provided by the authors of MASCOT [27] in SPDZ-2 [8]. We ran

Algorithm 2 $v \leftarrow \mathsf{Chi\text{-}squaredTest}(Nc, \mathbf{N}[Nc], [\![\mathbf{O}[4][Nc]]\!], t, [\![r]\!], [\![r']\!])$

Input: Nc: number of centers contributing data,
$\mathbf{N}[Nc]$: a table of size Nc containing the total sample size N_i of every center i,
$[\![\mathbf{O}[4][Nc]]\!]$: secret shared *observed* values of the 2×2 contingency table (mutation vs. disease), contributed by each of the Nc centers,
$t : \chi^2$ threshold value for the significance test,
$[\![r]\!], [\![r']\!]$: secret shared random values r and r'
Output: $v = 0$ or 1; $0 \rightarrow$ non-significant relationship between mutation, and disease, $1 \rightarrow$ significant relationship between mutation, and disease
for all P_j **do**
  {*Each party* P_j *engages in the protocol*}
  for all C_i **do**
    $[\![O_{k,l}]\!] \leftarrow [\![\sum_{i=1}^{Nc} O_{k,l}]\!]_i, k = 1, 2; l = 1, 2$
  end for
  $N \leftarrow \sum_{i=1}^{Nc} N_i$
  $[\![RT_i]\!] \leftarrow [\![O_{i,1} + O_{i,2}]\!], i = 1, 2$
  $[\![CT_i]\!] \leftarrow [\![O_{1,i} + O_{2,i}]\!], i = 1, 2$
  $[\![model_{k,l}]\!] \leftarrow [\![RT_k \cdot CT_l]\!], k = 1, 2, l = 1, 2$
  $[\![square]\!] \leftarrow [\![(N \cdot O_{1,1} - model_{1,1})^2]\!]$
  $[\![U_{i,j}]\!] \leftarrow [\![square \cdot model_{i,j}]\!], i = 1, 2; j = 1, 2$
  $[\![numerator]\!] \leftarrow [\![\sum_{i=1,j=1}^{2,2} U_{i,j}]\!]$
  $[\![denominator]\!] \leftarrow N \cdot [\![model_{1,1} \cdot model_{2,2}]\!]$
  $[\![difference]\!] \leftarrow [\![numerator]\!] - t \cdot [\![denominator]\!]$
  $[\![MaskedDifference]\!] \leftarrow [\![difference \cdot r + r']\!]$
  $MaskedDifference \leftarrow \mathsf{Open}([\![MaskedDifference]\!])$
  if $MaskedDifference > 0$ **then**
    return 1
  else
    return 0
  end if
end for

our experiments for timing the execution of our protocol on a desktop computer equipped with an Intel(R) Core(TM) i5-3570K processor, at 3.40GHz, with 16.00 GB RAM, and the Ubuntu 17.04 operating system.

We have only considered the online phase of the protocol, as the preprocessing is protocol-independent, and can be executed at any moment, well before the execution of the online phase. We note, however, that the offline phase is also practically efficient, and we refer the reader to MASCOT [27] for more details on the throughput of the offline phase. To give an indication of the cost of the offline phase of the protocol, we estimate the triple generation throughput, based on the experiment results presented by Keller et al. [27]. For three computation servers, equipped with eight-core i7 3.1 GHz CPU, and 32 GB RAM, in a local network with a 1 Gbit/s link per party, and a field $\mathbb{F}_p$, with p a 128-bit prime, approximately 2200 triples per second can be generated. Every time we recorded timings, before the execution of the online phase, we ran the setup script provided with the SPDZ-2 [8] software. This script simulates the offline phase, and creates all the necessary randomness for the execution of the online phase. The fact that the offline phase is simulated does not affect the performance, or efficiency of the online phase.

Our experiments were conducted on localhost with three computation servers. Hence, we do not take the network latency into account in the timing results we report. We do present the size of the data that each server has to send, as well as the communication rounds, and we consider this information to be sufficient for the reader to calculate the additional communication cost, based on the available network bandwidth.

For our performance analysis we have considered the following three scenarios: (1) the case where we calculate the numerator, and denominator of the $\chi 2$ statistic in the secret shared domain, and then open these two results; (2) the case where the secure comparison is implemented as described in Section 3; and (3) the case where we perform the secure comparison in the secret shared domain, and then open only the "True/False" answer to this question, as implemented in MASCOT. We selected these three scenarios, as the secure comparison is the most costly operation we need to carry out, and we wish to assess its impact on the performance of our protocol. Thus, with scenario (1) we completely avoid the secure comparison by opening the numerator, and denominator separately; with scenario (2) we perform the secure comparison using our randomization approach; and with scenario (3) we use the most popular method to carry out a secure comparison (see [9]), which is based on bit-decomposition –an inherrently inefficient approach–. Note that scenario (1) does not satisfy the security requirements of our application, and is presented only for the sake of performance comparison.

For all our timing results we have executed our protocol 10 times per case, and calculated the average execution time. The communication cost of the protocol is constant. For our experiments we have established that all input data is shared by one of the computation servers (namely Server 1), instead of the medical centers that would contribute the data in a real setting. This is reflected in the communication cost of the protocol for Server 1, which has to secret share all input data. Note, that in practice the secret sharing step can be pushed to an offline preprocessing phase.

In Table 5 we present the execution times of our approach, as well as the data sent by Server 1 (including the sharing of the original inputs), without performing a secure comparison (scenario 1). Server 1 is presented separately, because it has to do some extra tasks, such as sharing the inputs, collect all the final results, and print them, which is reflected in its execution times. The other two servers are grouped together, as their execution times are similar. Although we would expect the execution times to grow with the number of medical centers contributing data, this is not the case. This is because the execution times are so small that they can be highly affected by the computing environment. Furthermore, the communication cost of Server 1 includes also the secret sharing of the inputs. This is how all three scenarios have been executed. Recall, however, that the sharing of the inputs can be performed in a preprocessing phase, prior to the actual protocol execution, allowing the online phase to be less communication intensive. The communication cost for the other two servers is constant –1228 bytes–, since they do not share any inputs. The protocol completes its execution in 4 communication rounds, and consumes 9 triples, and 1 square from the preprocessing.

Table 5: Performance with No Secure Comparison

		Server 1		**Server *i*,** $i \neq 1$
Centers	**Patients**	**CPU Time**	**Sent Data**	**CPU Time**
20	200000	1.6 ms	4152 bytes	1.3 ms
40	400000	1.6 ms	6712 bytes	1.4 ms
60	600000	1.5 ms	9272 bytes	1.3 ms
80	800000	1.7 ms	11832 bytes	1.4 ms
100	1000000	1.7 ms	14392 bytes	1.4 ms

In Table 6 we provide the execution times of our alternative secure approach, which is based on randomizing the difference of the two numbers to be compared (scenario 2). Note that in the previously presented scenario (1) it is sufficient to work in a prime field of 128 bits, as the numbers we operate with never

grow larger than 114 bits. To perform the randomization of the difference of our numbers securely, however, we need to multiply them with a (random) number of roughly the same size. This implies that we need to work in a field of 256 bits to be able to handle the size of our quantities. More precisely, we calculated the largest that our quantities can grow in all cases, and we selected the random numbers to be upper bounded by these sizes. Our numbers can grow up to 100, 106, 110, 112, and 114 bits for 20, 40, 60, 80, and 100 medical centers, respectively. For this scenario, and the case of 20 centers, Server 1 has to send 82 (256-bit) elements (instead of 128-bit elements) to the two other servers. The two additional elements that have to be sent are the randomness r, and r', which is used to mask the difference that facilitates the execution of the secure comparison. The communication cost for Server 1 is analyzed in Table 6, while for the other two servers is constant and equal to 1632 bytes. The protocol completes its execution in 5 communication rounds, and consumes 10 triples, and 1 square from the preprocessing.

Table 6: Performance with Randomized Secure Comparison

		Server 1		**Server *i*,** $i \neq 1$
Centers	**Patients**	**CPU Time**	**Data Sent**	**CPU Time**
20	200000	1.7 ms	7616 bytes	1.4 ms
40	400000	1.7 ms	12736 bytes	1.5 ms
60	600000	1.9 ms	17856 bytes	1.7 ms
80	800000	2.1 ms	22976 bytes	1.8 ms
100	1000000	2.2 ms	28096 bytes	1.9 ms

In Table 7 we display the execution times of our protocol, using the standard, bit-decomposition based, secure comparison (scenario 3). Similarly to the second scenario, due to the size of our inputs (up to 114 bits), we need to scale our inputs representation to 256 bits field elements, so as to achieve adequate statistical security (always ≥ 40 bits). The protocol communication cost of Server 2, and 3 is constant –4244 bytes–, while for Server 1 it varies, based on the number of inputs it has to share, as shown in Table 7. The protocol completes its execution in 10 communication rounds, and consumes 50 triples, 1 square, and 31 bits from the preprocessing. As expected, this is the most inefficient of the three scenarios, both in terms of communication, and in terms of computational cost.

6 Conclusion

From the setup description of both suggested techniques, one can determine the first significant difference between them: in the homomorphic setting, the

Table 7: Performance with Standard Secure Comparison

		Server 1		Server i, $i \neq 1$
Centers	**Patients**	**CPU Time**	**Data Sent**	**CPU Time**
20	200000	2.2 ms	12712 bytes	1.9 ms
40	400000	2.3 ms	17832 bytes	2.0 ms
60	600000	2.3 ms	22952 bytes	2.0 ms
80	800000	2.5 ms	28072 bytes	2.2 ms
100	1000000	2.4 ms	33192 bytes	2.1 ms

medical centers only have to encrypt, and send their data to one party, namely the computation server; while for the multiparty computation they have to secret share their data with two or more computation servers. The execution times resulting from our experiments show that the MPC approach is significantly faster than the homomorphic approach. Even if we assume the encryption of the contingency tables by the medical centers to be part of a preprocessing phase, the homomorphic approach will take more than a second to complete its execution, while the computations in the MPC setup take only a few milliseconds. In terms of communication cost, the homomorphic setup has the advantage that it needs no communication during the computations. However, in terms of total amount of data that has to be transferred between the different parties, the MPC setup outperforms the homomorphic setup once more. We therefore recommend the MPC approach, as it is the most efficient out of the two approaches, and it does not rely on the strong assumption of semi-honest parties participating in the protocol.

Having compared the HE, and MPC approaches in a setting addressing the exact same problem, we have established that MPC can provide more efficient solutions with more relaxed security assumptions. Thus, we plan to proceed with future work on computing state-of-the-art statistics used in GWASs (instead of the more simple χ^2 test) in a privacy-preserving way, using MPC. To this end, we consider an interesting first step to study how we can express, or approximate logistic regression with low degree polynomials. Then, we can deploy MPC for computing them securely, which will yield solutions efficient enough to be used in practice.

Our work shows that, as long as we can express the statistics to be calculated with low-degree polynomials, privacy-preserving GWAS has become practical. We made the first step to efficient privacy-preserving GWAS with the secure calculation of the χ^2 test. Our solutions provide provable security guarantees, while being efficient for realistic sample sizes, and number of medical centers

contributing data to the studies. Interestingly, our solutions scale logarithmically in the number of subjects contributing data to the study, which means that as GWAS population sizes grow, our approach will remain suitable. We also propose a new masking-based comparison method, and show that in certain application scenarios, such as the GWAS scenario at hand, comparisons can be executed efficiently even in the HE setting, without leaking useful information about the underlying data.

Acknowledgements

This work was supported by the European Commission under the ICT programme with contract H2020-ICT-2014-1 644209 HEAT. Additionally, Yves Moreau, Jaak Simm, and Amin Ardeshirdavani were funded by Research Council KU Leuven: CoE PFV/10/016 SymBioSys; Flemish Government: IWT: Exaptation; O&O ExaScience Life Pharma; Exaptation, PhD grants Industrial Research fund (IOF): IOF/KP (Identification and development of new classes of immunosuppressive compounds and discovery of new key proteins involved in the T and B-cell activation); FWO 06260 (Iterative and multi-level methods for Bayesian multi-relational factorization with features); imec strategic funding 2017.

References

1. 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*, pages 341–356, Cham, 2016. Springer International Publishing.
2. Martin Albrecht. Complexity Estimates for Solving LWE. `https://bitbucket.org/malb/lwe-estimator/raw/HEAD/estimator.py`, 2000–2004.
3. Martin R. Albrecht, Rachel Player, and Sam Scott. On the Concrete Hardness of Learning with Errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.
4. Amin Ardeshirdavani, Erika Souche, Luc Dehaspe, Jeroen Van Houdt, Joris Robert Vermeesch, and Yves Moreau. NGS-Logistics: Federated Analysis of NGS Sequence Variants Across Multiple Locations. *Genome medicine*, 6(9):71, 2014.
5. Donald Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.
6. Charlotte Bonte, Carl Bootland, Joppe W. Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Faster Homomorphic Function Evaluation using Non-Integral Base Encoding. Cryptology ePrint Archive, Report 2017/333, 2017. `http://eprint.iacr.org/2017/333`.
7. Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In Martijn Stam, editor, *Cryptography and Coding 2013*, volume 8308 of *LNCS*, pages 45–64. Springer, 2013.

8. Bristol Crypto. SPDZ-2: Multiparty Computation with SPDZ Online Phase and MASCOT Offline Phase. `https://github.com/bristolcrypto/SPDZ-2`, 2016.

9. Octavian Catrina and Sebastiaan De Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer, 2010.

10. Hyunghoon Cho, David J Wu, and Bonnie Berger. Secure genome-wide association analysis using multiparty computation. *Nature biotechnology*, 36(6):547, 2018.

11. Scott D Constable, Yuzhe Tang, Shuang Wang, Xiaoqian Jiang, and Steve Chapin. Privacy-Preserving GWAS Analysis on Federated Genomic Datasets. *BMC medical informatics and decision making*, 15(5):S2, 2015.

12. Anamaria Costache, Nigel P Smart, Srinivas Vivek, and Adrian Waller. Fixed Point Arithmetic in SHE Schemes. In *SAC 2016*, LNCS. Springer, 2016.

13. CryptoExperts. FV-NFLlib. `https://github.com/CryptoExperts/FV-NFLlib`, 2016.

14. CryptoExperts, INP ENSEEIHT, and Quarkslab. NFLlib. `https://github.com/quarkslab/NFLlib`, 2016.

15. Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential Benchmarking based on Multiparty Computation. *IACR Cryptology ePrint Archive*, 2015:1006, 2015.

16. 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 *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

17. Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 643–662. Springer, 2012.

18. Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Manual for Using Homomorphic Encryption for Bioinformatics. Technical report, November 2015.

19. Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *International Conference on Machine Learning*, volume 48, pages 201–210. JMLR.org, 2016.

20. Shimon Even, Oded Goldreich, and Abraham Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, 28(6):637–647, 1985.

21. Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

22. Niv Gilboa. Two Party RSA Key Generation. In *Annual International Cryptology Conference*, pages 116–129. Springer, 1999.

23. Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V Pearson, Dietrich A Stephan, Stanley F Nelson, and David W Craig. Resolving Individuals Contributing Trace Amounts of DNA to Highly Complex Mixtures using High-Density SNP Genotyping Microarrays. *PLoS genetics*, 4(8):e1000167, 2008.

24. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending Oblivious Transfers Efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer, 2003.

25. Liina Kamm, Dan Bogdanov, Sven Laur, and Jaak Vilo. A New Way to Protect Privacy in Large-Scale Genome-Wide Association Studies. *Bioinformatics*, 29(7):886–893, 2013.

26. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively Secure OT Extension with Optimal Overhead. In *Annual Cryptology Conference*, pages 724–741. Springer, 2015.

27. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.

28. Miran Kim and Kristin Lauter. Private Genome Analysis through Homomorphic Encryption. 15:S3, 12 2015.

29. Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private computation on encrypted genomic data. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014*, pages 3–27, Cham, 2015. Springer International Publishing.

30. Kristin Lauter, Adriana López-Alt, and Michael Naehrig. *Private Computation on Encrypted Genomic Data*, pages 3–27. Springer International Publishing, Cham, 2015.

31. Wen-Jie Lu, Yoshiji Yamada, and Jun Sakuma. Privacy-Preserving Genome-Wide Association Studies on Cloud Environment using Fully Homomorphic Encryption. 15:S1, 12 2015.

32. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. *J. ACM*, 60(6):Art. 43, 35, 2013.

33. Bradley Malin. Re-identification of familial database records. In *AMIA annual symposium proceedings*, volume 2006, page 524. American Medical Informatics Association, 2006.

34. Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can Homomorphic Encryption be Practical? In Christian Cachin and Thomas Ristenpart, editors, *ACM Cloud Computing Security Workshop – CCSW*, pages 113–124. ACM, 2011.

35. Karl Pearson. X. On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is such that it can be Reasonably Supposed to have Arisen from Random Sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.

36. Michael O Rabin. How to Exchange Secrets with Oblivious Transfer. Technical report, Aiken Computation Lab, Harvard University, 1981.

37. Md Nazmus Sadat, Md Momin Al Aziz, Noman Mohammed, Feng Chen, Shuang Wang, and Xiaoqian Jiang. SAFETY: Secure GWAS in Federated Environment through a Hybrid Solution with Intel SGX and Homomorphic Encryption. *CoRR*, abs/1703.02577:1–17, 2017.

38. Sean Simmons and Bonnie Berger. Realizing Privacy Preserving Genome-Wide Association Studies. *Bioinformatics*, 32(9):1293–1300, 2016.

39. The Global Alliance for Genomics and Health. A Federated Ecosystem for Sharing Genomic, Clinical Data. *Science*, 352(6291):1278–1280, jun 2016.

40. Caroline Uhlerop, Aleksandra Slavković, and Stephen E Fienberg. Privacy-Preserving Data Sharing for Genome-Wide Association Studies. *The Journal of privacy and confidentiality*, 5(1):137, 2013.

41. Rui Wang, Yong Fuga Li, XiaoFeng Wang, Haixu Tang, and Xiaoyong Zhou. Learning your identity and disease from research papers: Information leaks in genome wide association study. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 534–544, New York, NY, USA, 2009. ACM.

42. Yihua Zhang, Marina Blanton, and Ghada Almashaqbeh. Secure Distributed Genome Analysis for GWAS and Sequence Comparison Computation. *BMC medical informatics and decision making*, 15(5):S4, 2015.

43. Yuchen Zhang, Wenrui Dai, Xiaoqian Jiang, Hongkai Xiong, and Shuang Wang. Foresee: Fully outsourced secure genome study based on homomorphic encryption. *BMC Medical Informatics and Decision Making*, 15(5):S5, Dec 2015.

Chapter 8

EPIC: Efficient Private Image Classification (or: Learning from the Masters)

Publication data

E. Makri, D. Rotaru, N. P. Smart, F. Vercauteren. "EPIC: Efficient Private Image Classification (or: Learning from the Masters)" In *Cryptographers' Track at the RSA Conference* Springer, Cham., 2019, pp. 473-492.

EPIC: Efficient Private Image Classification (or: Learning from the Masters)

Eleftheria Makri[1,2], Dragos Rotaru[1,3], Nigel P. Smart[1,3], and Frederik Vercauteren[1]

[1] imec-COSIC, KU Leuven, Belgium.
[2] ABRR, Saxion University of Applied Sciences, The Netherlands.
[3] University of Bristol, UK.
firstname.lastname@esat.kuleuven.be

Abstract. Outsourcing an image classification task raises privacy concerns, both from the image provider's perspective, who wishes to keep their images confidential, and from the classification algorithm provider's perspective, who wishes to protect the intellectual property of their classifier. We propose EPIC, an efficient private image classification system based on support vector machine (SVM) learning, secure against malicious adversaries. EPIC builds upon transfer learning techniques known from the Machine Learning (ML) literature and minimizes the load on the privacy-preserving part. Our solution is based on Secure Multiparty Computation (MPC), it is 34 times faster than Gazelle (USENIX 2018) –the state-of-the-art in private image classification– and it improves the communication cost by 50 times, with a 7% higher accuracy on CIFAR-10 dataset. For the same accuracy as Gazelle reaches on CIFAR-10, EPIC is 700 times faster and the communication cost is reduced by 500 times.

1 Introduction

Visual object recognition is an important machine learning application, deployed in numerous real-life settings. Machine Learning as a Service (MLaaS) is becoming increasingly popular in the era of cloud computing, data mining, and knowledge extraction. Object recognition is such a machine learning task that can be provided as a cloud service. However, in most application scenarios, straightforward outsourcing of the object recognition task is not possible due to privacy concerns. Generally, the *image holder* who wishes to perform the image classification process, requires their input images to remain confidential. On the other hand, the *classification algorithm provider* wishes to commercially exploit their algorithm; hence, requires the algorithm parameters to remain confidential.

We consider an approach, which facilitates the outsourcing of the image classification task to an external classification algorithm provider, without requiring

the establishment of trust, contractually or otherwise, between the involved parties. We focus on the evaluation task (i.e., labeling a new unclassified image), and not the learning task. Our proposal is based on Secure Multiparty Computation (MPC), and allows for private image classification without revealing anything about the private images of the image holder, nor about the parameters of the classification algorithm. Unlike previous work [4,5,20], we can fully outsource the task at hand, in such a way that the classification algorithm provider does not need to be the same entity as the *cloud computing provider*. Although any of the involved parties (i.e., the classification algorithm provider, and the image holder) can play the role of one of the MPC servers, this is not a requirement for guaranteeing the security of our proposal. MPC allows distribution of trust to two or more parties. As long as the image holder (resp. the classification algorithm provider) trusts at least one of the MPC servers, their input images (resp. their classification algorithm parameters) remain secret.

MPC allows a set of mutually distrusting parties to jointly compute a function on their inputs, without revealing anything about these inputs (other than what can be inferred from the function output itself). Currently, MPC allows one to compute relatively simple functions on private data; arbitrarily complex functions can be supported, but with an often prohibitive computational cost. EPIC, our privacy-preserving image classification solution, combines the techniques of transfer learning feature extraction, support vector machine (SVM) classification, and MPC. In this work we use recently developed techniques for generic image classification (within the ImageNet competition) such as transfer learning to extract powerful generic features. Transfer learning using raw CNN features has been studied extensively by Azizpour et al. [3], and Yosinski et al. [55]. Then, the computation done in the MPC setting is minimized to only evaluate a simple function with secret shared inputs.

We focus on classification via SVM, as opposed to using more sophisticated techniques, such as Neural Networks (NNs), in the privacy-preserving domain to minimize the computational cost. While the field of private image classification is shifting towards NN-based approaches [25,33,43], we show that it is not necessary to use private NNs, as we can achieve classification with better accuracy by using generic NNs to improve the feature extraction techniques used. Although Convolutional NNs (CNNs) are the state-of-the-art for image classification [23], we confirm that SVMs can achieve high accuracy, as long as they are provided with good quality features. Transforming a NN to a privacy-preserving one results in inefficient solutions (e.g., 570 seconds for one image classification by CryptoNets [20]).

A schematic representation of the application scenario treated by EPIC is given in Fig. 1. Using additive secret sharing techniques both the *classification*

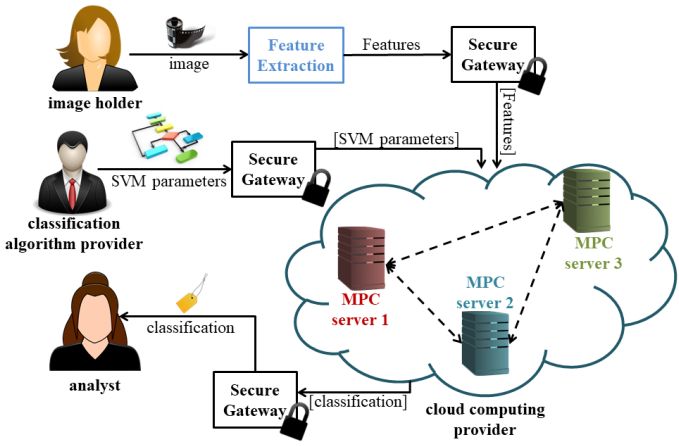


Fig. 1. A schematic representation of the private image classification scenario.

algorithm provider, and the *image holder* share their inputs to the $n \geq 2$ *MPC servers*. Note that no information about the actual secret inputs can be gained by the individual shares alone. Thus, each MPC server learns nothing about the inputs of the two parties. The cluster of the MPC servers comprise the *cloud computing provider*, which together execute the MPC protocol to produce the final classification result. The MPC servers communicate via authenticated channels to accomplish what the protocol prescribes. The protocol completes its execution by having all MPC servers sending their share of the final classification result to the designated party, who can then reconstruct the result by combining the received shares. This party can be the *image holder*, or an external *analyst*, assigned to examine the classification results, without getting access to the underlying private images. The involved parties (image holder, classification algorithm provider, and –potentially– analyst), may play the role of the MPC servers themselves, avoiding completely the outsourcing to the cloud provider(s).

A key aspect of our work is how the data is processed before the MPC engine is used to perform the classification. The SVM classification is performed on so-called feature vectors, and not directly on the images. The way one determines these feature vectors not only affects accuracy, but it also has an impact on security. As shown in Fig. 1 the image holder performs the feature extraction on the input image before it is passed to the secure gateway. Thus this feature extaction must not be specific to the algorithm classification provider; otherwise the extracted features could reveal information about exactly what is being

classified. We apply a generic feature extraction method, which is independent of the underlying classification task.

In particular, we employ TensorFlow [1], to extract features based on the activation of a deep CNN (specifically the Inception-v3 [47] CNN) trained on a set of object recognition tasks, *different* from the target task. This method is known as CNN-off-the-shelf in the ML literature, and it has been successfully applied in various image recognition tasks [15,42]. Since the CNN is generic, it can be released in the clear, and hence become part of the image holder's preprocessing. This not only gives us a security benefit, but it also significantly improves the accuracy of our method. There are many public CNNs available online for generic feature extraction in Caffe's Model Zoo, which can be used with our EPIC solution to add a privacy dimension to a typical ML problem [24]. In our paper we selected Inception-v3 as the public CNN to extract features, because it suits many generic image recognition tasks, and allows us to benchmark EPIC against previous solutions on traditional datasets such as CIFAR-10.

We also present a second variant of EPIC, which aims at allowing a tradeoff between the accuracy of the classifier's predictions, and its performance. It does so by deploying a kernel approximation method, on top of Inception-v3 features for dimensionality reduction.

We implemented our solution using SPDZ [7], which was introduced by Damgård et al. [12,13], it is based on additive secret sharing, and it is proven secure in the active security model, in the full versions of the papers [12,13]. We assume the reader to be familiar with MPC, but we discuss preliminaries on the techniques used by EPIC for completeness in the full version of our paper [34]. EPIC outperforms the state-of-the-art in secure neural network inference [25], both in terms of efficiency, and in terms of prediction accuracy. Our implementation shows that privacy-preserving image classification has become practical. As shown in Table 1, we are the only provably secure work in the active security model, which is a property we inherit by the chosen implementation frameworks. A system like EPIC could find application in numerous real-life cases, such as in purchase scenarios where visual insepction is performed, or when targeted surveillance is required without compromising non-targets' privacy.

Our contributions are thus four fold: i) We enable full outsourcing of privacy-preserving image classification to a third independent party using a simple technique yet much faster and accurate than others, which require complicated machinery. ii) Our solution does not leak *any* information about the private images, nor the classifier, while being the first to provide active security. iii) We show how to deploy a data-independent feature extraction method to alleviate the privacy-preserving computations, while increasing accuracy and efficiency. iv)

We demonstrate the practicality of our approach, both in terms of efficiency, and in terms of accuracy, by conducting experiments on realistic datasets.

2 Related Work

Privacy-preserving machine learning can focus either on providing a secure training phase, a secure classification phase, or both secure training and classification phases. The first research works in the field aimed at designing a privacy-preserving training phase. Recently, due to the advent of cloud computing, and Machine Learning as a Service, more and more works focus on the design of a privacy-preserving classification phase. Fewer works have attempted to address both the training, and the classification phases in a privacy-preserving manner.

To facilitate an easy comparison of the related work, we summarize the main features of each proposal in Table 1.

- The first column of Table 1 is the reference to the corresponding paper.
- The second column indicates whether the work considers secure training (T), secure training and classification (T+C), or only secure classification (C).
- The third column indicates the security model, under which the proposed protocols are secure, where P stands for passive security, and A stands for active security. N/A (not applicable) refers to differential privacy techniques, which are designed to protect against inference about the inputs from the outputs, and thus are orthogonal to the issue of securing the computation which we deal with.
- The fourth column denotes the method used to preserve privacy. DP stands for differential privacy; SP stands for selective privacy, and refers to the unique characteristic of the work of Shokri and Shmatikov [46] allowing the users to decide how much private information about their learned models they wish to reveal. SHE stands for Somewhat Homomorphic Encryption, 2-PC for 2-Party Computation, and MPC stands for Multiparty Computation (which could include 2-PC).
- The fifth column lists the training method(s) used. N-L SVM stands for non-linear SVM, NN for Neural Networks, LM for Linear Means, FLD for Fisher's Linear Discriminant, HD for hyperplane decision, LIR for linear regression, LOR for logistic regression, and DT for decision trees.
- The sixth column lists the information that is revealed by the protocol execution. C stands for information about the classifier, and TD for information about the training data. We note with boldface letters the information that is intentionally revealed by the protocol execution, and we mark with an asterisk the information that is protected by means of differential privacy techniques.

	Func.	Sec. model	Privacy mthd	Train mthd	Info leak	Impl.
[30]	T	N/A	DP	N-L SVM	**C**; TD*	✓
[31]	T	N/A	DP	N-L SVM	**C**; TD*	✓
[46]	T	P	SP	NN	**C**	✓
[51]	T	P	MPC	N-L SVM	**C**	✓
[48]	T	P	MPC; DP	N-L SVM	**C**	✓
[9]	T	P	MPC; DP	NN	**C***	✓
[21]	T+C	P	SHE	LM; FLD	no	✓
[2]	T+C	P	SHE	Bayes; random forests	no	✓
[29]	T+C	P	2-PC	N-L SVM	no	×
[10]	T+C	P	2-PC	NN	TD	×
[35]	T+C	P	2-PC	NN; LIR; LOR	no	✓
[20]	C	P	SHE	NN	no	✓
[4]	C	P	SHE	N-L SVM	C	✓
[8]	C	P	SHE	NN	no	×
[6]	C	P	SHE; 2-PC	HD; Bayes; DT	no	✓
[41]	C	P	2-PC	N-L SVM	no	✓
[5]	C	P	2-PC	NN	**C**	×
[36]	C	P	2-PC	NN	no	✓
[45]	C	P	2-PC	NN	no	✓
[33]	C	P	2-PC	NN	**filter size**	✓
[43]	C	P	2-PC	NN; SVM	no	✓
[25]	C	P	2-PC	NN	no	✓
EPIC	C	A	MPC	SVM	no	✓

Table 1. Comparison of the related work.

Information that can potentially, and unintentionally be leaked is noted with normal, non-boldface letters.

- The last column indicates whether the work provides an implementation.

Training a SVM in a privacy-friendly way, has been previously considered based on techniques of differential privacy [30,31]. Despite the little overhead that these techniques incur, which makes them competitive from an efficiency perspective, they do not consider the security of the actual *computation* during the training or classification. Shokri and Shmatikov [46] achieve such privacy-preserving collaborative deep learning with multiple participants, while refraining from using cryptographic techniques. Their work focuses on learning the NN, but they also consider protecting the privacy of each individual's NN, allowing the participants to decide how much information to share about their models.

A lot of research has been devoted to provable privacy-preserving techniques for training a classifier. Privacy-preserving data mining has been an active re-

search area since the seminal work of Lindell and Pinkas [32]. More recently, Vaidya et al. [51] showed how to train a SVM classifier, in a privacy-preserving manner, based on vertically, horizontally, and arbitrarily partitioned training data. In follow-up work, Teo et al. [48] improved upon the efficiency of the solution of Vaidya et al. [51], and showed that their approach scales well to address the challenges of data mining on big data. Chase et al. [9] combine MPC techniques with differential privacy techniques to achieve private neural network learning. Their work provides provable security guarantees for the learning phase (in the passive security model), and adds noise to the resulting network to protect its privacy during classification.

A parallel research line aiming to address the same challenge, namely privacy-preserving data mining, is based on homomorphic encryption (instead of MPC). The notion of homomorphic encryption dates back to the work of Rivest et al. [44], but only recently fully homomorphic encryption was devised [19]. This type of homomorphic encryption allows the computation of any polynomial function on the encrypted data, and unlike MPC, it does not require communication, as the task can be outsourced to one single party. Since the seminal work of Gentry [19], a lot of somewhat homomorphic encryption schemes have been proposed, allowing computations of polynomial functions of a limited degree. Graepel et al. [21] consider both machine learning training, and classification based on encrypted data, with their solutions being secure in the passive model. Due to the selected homomorphic encryption scheme, Graepel et al. [21] cannot treat comparisons efficiently, which excludes SVM-based solutions. Addressing both learning, and classification based on extremely random forests, and naïve Bayes networks, Aslett et al. [2], also work on homomorphically encrypted data.

One of the first private SVM classifiers was proposed by Laur et al. [29], which addresses both the training and the classification in a privacy-preserving manner. Their work combines the techniques of homomorphic encryption, secret sharing, and circuit evaluation, into a passively secure 2-PC solution. Concurrently, and independently Dahl [10] is working on using the same MPC framework as in our work, to realize both the training, and the classification of CNN based privacy-preserving algorithms. While Dahl [10] is deploying CNNs instead of SVM, he needs to apply them in a non-black-box fashion. The protocol of Dahl [10] allows some leakage of information during the training phase, which is not the case with our approach. SecureML [35] also considers both training and classification in the 2-PC setting, and the passive security model. These approaches [29,10,35] can only treat the two-party setting, and cannot be trivially extended to allow the classifier provider to be a different entity than the cloud provider.

Other works focus particularly on the private image classification problem, instead of the training of the model. Gilad-Bachrach et al. [20] propose a solu-

tion applicable to the image classification problem, based on homomorphically encrypted data. The resulting *CryptoNets* [20] provide an accuracy of 99% for the MNIST dataset, and can make on average 51739 predictions per hour. However, this is only the case when the predictions are to be made simultaneously; for a single prediction the task takes 570 seconds to complete.

Recent work by Barnett et al. [4] demonstrated the potential of polynomial-kernel SVM to be used for classification in a privacy-preserving manner. Specifically, Barnett et al. apply SVM techniques for the classification –as in our work– but on encrypted data. Although they mention the potential of an MPC approach to be more efficient in this setting, they do not consider it, because direct translation of the protocols to MPC would require interaction between the client and the classification algorithm provider during the computations. We overcome this limitation by extending the application scenario in a way that allows the classification task to be fully outsourced to a cluster of independent third parties. We implement their approach using SPDZ in a more secure way by keeping the PCA components private (they choose to make them public). This implementation is more expensive than EPIC, due to the non-linearity of the polynomial SVM, and it is also less accurate. Albeit inefficient and inaccurate, it provides an initial benchmark, and it shows the gap between an FHE and an MPC approach (see details in Section 4). Chabanne et al. [8] attempted to approximate commonly used functions used in NN-based classification in a SHE-friendly manner. Despite the high prediction accuracy that their work achieves, Chabanne et al. do not provide any performance evaluation results.

In the 2-PC setting, Bost et al. [6], and Rahulamathavan et al. [41] focus on the problem of private classification, where both the classifier parameters, and the client's input to be classified need to remain private. The latter approach does not consider linear SVM, while both approaches only offer passive security. Barni et al. [5] propose private NN-based data classification, also in the 2-PC setting and passive security model. They suggest three protocols, which offer different privacy guarantees for the classifier owner, while always protecting fully the client's input. Follow up work by Orlandi et al. [36] extends the work of Barni et al. in terms of privacy. DeepSecure [45] is another work in the 2-PC setting, and the passive security model, using Garbled-Circuit techniques. A direct performance comparison of DeepSecure versus CryptoNets [20] confirmed a significant efficiency improvement achieved by DeepSecure.

The recently proposed MiniONN [33] is one of the latest NN-based data classification approaches in the 2-PC setting. MiniONN demonstrates significant performance increase compared to CryptoNets, without loss of accuracy; as well as better accuracy compared to SecureML [35], combined with increased performance. However, it still operates in the 2-PC setting, which is more restricted

than the MPC setting we consider, and it only offers passive security. Under a comparable configuration as MiniONN, and still in the passive security model, Chameleon [43] achieves a 4.2 times performance improvement. Chameleon operates in the 2-PC setting, under the assumption that a Semi-Honest Third Party (STP) is engaged in the offline phase to generate correlated randomness. Despite the strong STP assumption, Chameleon does not need the third party for the online phase, while it gets a significant performance increase from this STP.

Gazelle [25], the latest work on secure NN classification, outperforms, in terms of efficiency, the best previous solutions in the literature [20,33,43], by carefully selecting which parts of the CNN to carry out using a packed additively homomorphic encryption, and which using garbled circuits. EPIC performs better than Gazelle, while also being secure in the active security model. This is because EPIC only treats linear computations in the privacy-preserving domain.

To the best of our knowledge, we are the first to provide a privacy-preserving image classification tool combining SVM classification with transfer learning feature extraction, offering active security. EPIC is more efficient than previous work and achieves prediction accuracy higher than that of the related work on the same datasets, although it does not deploy sophisticated NN-based classification on the private inputs. Interestingly, EPIC is not limited to the 2-PC setting, allowing a broad range of application scenarios to be treated by our solution.

3 EPIC

The proposed private image classification solution, EPIC, is based on transfer learning techniques [49] for feature extraction. The EPIC algorithm for image classification runs in two phases. In the first phase the image is passed through a generic feature extraction method. Being generic, i.e., not task specific, this method can be published in-the-clear and hence can be applied by the image holder before passing the output securely to the MPC engine. In the second phase the actual classification, via an SVM, is applied. This SVM is specific to the task at hand, and hence needs to be securely passed to the MPC engine. We thus have two problems to solve Feature Extraction and SVM classification.

Feature Extraction. High quality features are key to the accuracy of a trained classifier. We ensure high quality feature extraction by deploying the techniques of transfer learning. Specifically, we perform feature extraction based on Inception-v3 [47], which is a *public* CNN classifier trained on a set of non-privacy-sensitive object recognition tasks. Commonly, the training for such a CNN classifier is performed on large datasets, which enhances the prediction accuracy of the classifier. In our context, the trained classifier extracts features based on the

activation of a deep convolutional network. Our work shows that powerful feature extraction is essential to the quality of the final classification accuracy. In fact, we demonstrate that the high-complexity (CNN) tasks can be learned on non-private datasets, and still use their power for feature extraction of unrelated tasks. Eventually, this allows us to deploy only linear functions for the actual classification, which enables accurate, and efficient privacy-preserving solutions.

SVM Classification. Despite the increasing popularity and high effectiveness of CNN classification techniques, the direct deployment of CNN techniques requires large training datasets [14] that are potentially difficult to obtain when the underlying data is privacy sensitive. In addition, black-box transformation of CNN-based methods to their privacy-preserving equivalents will result in classifiers that are computationally prohibitive to use. Thus using a light-weight classification method such as SVMs can be beneficial in privacy sensitive environments, and their evaluation can be done (as we show) in a secure manner. With the CNN features, an SVM can learn quickly from very few positive examples, which shows that they are useful to perform one-shot learning [15]. Thus, we opted for the design of a private SVM classifier, while using the techniques of CNN-based *transfer learning* in the context of feature extraction, which does not raise privacy concerns.

To classify a new unlabeled input with our classifier trained with a linear SVM, we need to securely evaluate the following equation:

$$\mathsf{class}(\mathbf{h}) = \arg\max_i(\mathbf{x_i} \cdot \mathbf{h} + b_i), \tag{1}$$

where:

- $\mathbf{h}$ is the vector representing the client's image, and has been provided to the MPC servers in shared form;
- b_i is the model intercept (aka bias), calculated by the classification algorithm provider during the learning phase and secret shared to the MPC servers;
- $\mathbf{x_i}$ are the n support vectors.

The support vectors $\mathbf{x_i}$, and the model intercepts b_i are assumed to need protection, as they represent the intellectual property of the learned model.

Feature Reduction. To achieve efficient training of kernel machines (such as SVM) aimed at non-linear problems, several approximation methods (e.g., the method of Rahimi and Recht [39]) have been proposed. Such approaches have the goal to alleviate the (cleartext) computational, and storage cost of the training, incurred by the high dimensionality of the data, especially when the training datasets are large. The approximation generally is implemented by mapping the

input data to a low-dimensional feature space, such that the inner products of the mapped data are approximately equal to the features of a more complex (e.g., Gaussian) kernel. This is known as the kernel trick. These features are then combined with linear techniques (e.g., linear SVM), yielding an efficient training, but also an efficient classification, which we are able to implement in a privacy-preserving way.

One of the first successful approaches for kernel approximations, achieving high accuracy, was proposed by Rahimi and Recht [39], and is based on random features, which are independent of the training data. To the contrary, Nyström based kernel approximations [53,16], are data dependent. Although Nyström approximations outperform randomly extracted features [54] in terms of accuracy, being data dependent makes them unfit for our purposes, as they require applying non-linear functions on the private inputs. From a computational, and storage efficiency perspective, data independent approximations are favored.

We discovered that a variant of the method proposed by Rahimi and Recht [40] is presented in the scikit-learn package [37]. This implements an RBF (Radial Basis Function) sampler, which allows to transform the features without using the training data. This dimensionality reduction (like the feature extraction) is deployed both for the training, and for the data classification. Since the feature selection is random (i.e., data independent), it can be performed on the cleartext data, both by the classification algorithm provider, and by the client, without raising privacy concerns.

Our second variant of EPIC (see below) supports dimensionality reduction for free, by placing all the computational load on the cleartext. This variant makes use of an algorithm implementing the RBF sampler listed in Fig. 2. In our application scenario, the algorithm provider broadcasts the RBF sampler parameters, namely the γ parameter and the feature size. The γ parameter does not reveal any information about the dataset. Note that γ is a floating point number, which is varied to match a cross-validation score on the training data. The shape variable is the feature size of a point (set to 2048), which is the output of Inception-v3.

EPIC – Simple Variant: The classification algorithm provider has already trained their SVM classifier. The parameters for the SVM classification are shared to the MPC servers by the classification algorithm provider and are never revealed to the image holder (nor the analyst). The image holder applies the Inception-v3 feature extraction to their image, and takes the next to last layer, which has a feature size of 2048, as their output. The resulting features are then shared (via the secure gateway) to the MPC servers by the image holder, and thus are kept secret from the classification algorithm provider. We indicate se-

Scikit-learn variant of Random Kitchen Sinks [37].

Init: Set $\gamma, \mathsf{shape}, \mathsf{state}_{\mathsf{random}}, \mathsf{nc}$.

Fit:

1. Select $\mathsf{weights}_{\mathsf{random}} = \sqrt{2 * \gamma} \cdot \mathsf{state}_{\mathsf{random}}.\mathcal{N}(0, 1)$ of size $\mathsf{nc} \times \mathsf{shape}$, with mean 0 and standard deviation 1.
2. Assign $\mathsf{offset}_{\mathsf{random}} = \mathsf{state}_{\mathsf{random}}.\mathcal{U}(0, 2 \cdot \pi)$ of size nc.

Transform(x):

1. $\mathsf{projection} = x \cdot \mathsf{weights}_{\mathsf{random}} + \mathsf{offset}_{\mathsf{random}}$.
2. $\mathsf{projection} = \cos(\mathsf{projection})$.
3. $\mathsf{projection} = \mathsf{projection} \cdot (\sqrt{2}/\sqrt{\mathsf{nc}})$

Fig. 2. RBF Sampler.

cret shared data in square brackets (Fig. 1). The MPC engine then evaluates the SVM securely on the features and outputs the result to the analyst (or image holder).

Note that although EPIC does not allow any information leakage about the private SVM parameters, recent work by Tramer et al. [50] showed that only black-box access to the classifiers can still serve to recover an (near-)equivalent model. We consider this problem to be out of this work's scope, as it can easily be tackled by restricting the number of queries an external party is allowed to perform on the MPC Engine. This type of attacks has not been averted by any of the secure computation solutions in the related work.

EPIC – Complex Variant: The second variant of EPIC protocol is summarized in Fig. 3. This EPIC variant trades a small percentage of the classification accuracy to increase efficiency. It achieves this tradeoff by deploying the *kernel approximation* dimensionality reduction explained above, and in particular the kernel approximation sub-step is also considered to be part of the feature extraction phase. Here the algorithm provider needs to publish the feature size of a point (in our case 2048) and the parameter γ from above. At first sight it might seem that γ reveals information about the training data, but we noticed that for our datasets one can fix $\gamma = 2^{-13}$ to a small value and modify the regularization parameter C of the SVM. This parameter C will always remain private to the algorithm provider, hence there is no information leakage. We stress again that

for both cases, the CNN feature extraction is input independent, so privacy is maintained for the image holder and algorithm provider.

Specifically, the protocol starts with the Setup phase, where the algorithm provider (AP) performs the kernel approximation (from Fig. 2) on its own dataset, and broadcasts the type of CNN used, and the Init parameters necessary for the feature reduction at the image holder (IH) side. Then, the algorithm provider secret shares the SVM parameters to the MPC Engine (Eng). Secret shared values are denoted in double square brackets. In the evaluation phase, the image holder performs the feature extraction locally (given the previously obtained parameters), and secret shares the new point to be classified by Eng. Then, the MPC protocol computes Eq. 1.

EPIC Protocol with kernel approximation as feature reduction

Setup:

1. Algorithm Provider (AP) broadcasts the type of CNN used for feature extraction.
2. AP computes γ from Fig.2 on its own training data. Then AP broadcasts the Init variables from Fig. 2 and secret shares the support vectors x_i, b_i to the MPC engine (Eng). These are stored on Eng as $[\![x_i]\!], [\![b_i]\!]$.

Evaluate:

1. Image Holder (IH) uses public CNN to extract features h' from its image. Then IH maps $h' \mapsto h$ locally using the RBF sampler initialized with the γ broadcasted by AP to obtain a smaller number of features. The new point h is further secret shared to the Eng and stored as $[\![h]\!]$.
2. Eng uses $[\![x_i]\!], [\![b_i]\!], [\![h]\!]$ to compute Eq. 1 with a shared result: $[\![\mathsf{class}(\mathbf{h})]\!]$.

Fig. 3. Protocol for SVM classifcation with RBF sampler.

4 Experiments

Experimental Setup. Our experiments are conducted on two MPC servers, which yields the most efficient solution, but we also show how the proposed system scales with more than two MPC servers. We assume a protocol-independent, input-independent preprocessing phase that takes place prior to the protocol execution between the MPC servers. The inputting parties do not need to be aware, nor contribute to this phase. The preprocessing creates the randomness needed to boost the efficiency of the online phase, and allows the inputting par-

ties (image holder and classification algorithm provider) to securely share their inputs.

The online phase begins with the image holder, and the algorithm provider sharing their inputs (reduced CNN features, and SVM parameters, resp.) to the MPC servers. This is performed by executing an interactive protocol between each inputting party and the two MPC servers, as Damgård et al. [11] proposed. Then, the actual private image classification task is executed only between the two MPC servers, as in the **Evaluate** phase of Fig. 3. In the end, each MPC server sends their resulting share to the image holder, or the analyst, who can combine the shares and reconstruct the cleartext result, which is the desired class label.

From Fixed Point Arithmetic to Integers. For the secure integer comparison sub-protocols that EPIC deploys, we selected the statistical security parameter to be $\kappa = 40$ bits. We stress that everywhere the computational security parameter is set to $\lambda = 128$. We observed experimentally after running the scikit-learn's RBF (see Fig. 2) on top of Inception-v3 that each feature is bounded by $\mathsf{abs}(x_i) \leq 15$ where $\mathsf{len}(x) \leq 2048$.

To avoid the costly fixed point arithmetic, we scale each feature x_i by a factor f, and then perform arithmetic on integers. Particularly, we compute $x_i \cdot f$ and then round it to the nearest lower integer. We varied f, and evaluated the SVM's accuracy. We experimentally concluded that setting $f = 2^8$ gives sufficient accuracy, as if working on floating point numbers, while lowering the scale factor f decreased the accuracy by more than 1%. If $f = 2^8$ then to compute a class score from Eq. 1 becomes: $s = \sum_{j=1}^{2048}(2^8 \cdot x_{ij} \cdot 2^8 \cdot h_j) + 2^{16} \cdot b_i$ since we need to scale both support vectors x_i as well features h. Using the fact that each component is bounded by 15 then clearly $s \leq 2^{35}$.

To improve the underlying MPC performance we wanted to aim for using a 64-bit prime modulus for the underlying linear secret sharing scheme. Unfortunately, if our inputs are of 35 bit size then there is no room left to perform the secure comparisons in arg max with 40 bits statistical security, as $35 + 40 > 64$. Surprisingly, for all the datasets we experimented with, s was less than 20 bits long, because some of the x_{ij}'s are negative. Hence, we could run everything on 64-bit primes with 40-bit statistical security, while ensuring there is no information leak from the comparisons. We can achieve an even tighter bounding by normalizing the features using the L2-Norm, after the RBF-Sampler invocation. In our setting this is not necessary, since the expected bound on s is already low (20 bits). We also experimented (see later) with higher statistical security of 100 bits by using 128-bit prime fields.

For the feature reduction we considered whether to use RBF or PCA, and concluded that RBF is more suitable for our purposes. Despite the accuracy loss that RBF incurs compared to PCA, it is justified to use RBF for reasons of computational and communication efficiency. For a more detailed comparison between RBF and PCA feature reduction in our setting, we refer the reader to the full version of our paper [34].

Datasets. We selected three image datasets: CIFAR-10, MIT-67, and Caltech-101, to show how EPIC scales in terms of performance, when increasing the number of classes, and to illustrate its classification accuracy.

- **CIFAR-10** [28]: This is a dataset of 60000 32x32 color images, out of which 50000 are training images and 10000 are test images. CIFAR-10 features 10 classes of objects, with 6000 images per class. The accuracy metric is the quotient between correctly classified samples and total number of samples.
- **MIT-67** [38]: MIT-67 has 15620 indoor images from 67 scene categories. We used 80 images per class for training, and the rest of the pictures for testing. The accuracy metric used here is the mAP (mean Accuracy Precision), which consists of calculating the average over the accuracies of each class.
- **Caltech-101** [17]: This dataset contains pictures of objects of 102 categories. Each class has at least 31 images and we chose to use 30 images from each class for the training. The accuracy metric is mAP, just as in MIT-67.

Training. We trained the SVM on the cleartext versions of the aforementioned datasets. Feature extraction was done after resizing each image to 256x256. We trained Linear SVMs based on the one-versus-all strategy (OvA) [52], because it is more efficient to evaluate n classifiers in MPC instead of $n(n-1)/2$. Note that we chose to avoid the data augmentation trick, and adopted the training method presented in DeCAF [15] using the original datasets, and raw features from Inception-v3 [15]. To find parameters that yield high classification accuracy, we have done a grid search for the γ required in the RBF, and the parameter C, which denotes the size-margin hyperplane for the SVM decision function.

We stress that EPIC achieves a sufficient classification accuracy. Given that EPIC workings have been purposely kept simple to allow for efficient secure computations, we consider the classification accuracy of EPIC comparable to that achieved by the state-of-the-art (non-privacy-preserving) works in the ML community. The best classification accuracy in-the-clear on the CIFAR-10 dataset is 97.14% [18], while EPIC achieves 88.8%. On the MIT-67 dataset, EPIC achieves 72.2% accuracy, while the state-of-the-art in-the-clear solution [27] reports an accuracy of 83.1%. More interestingly, on Caltech-101, the state-of-the-art accuracy in-the-clear is still 93.42% [22], while EPIC achieves 91.4%.

Classification Accuracy and Performance Evaluation. We executed our experiments, simulating the two MPC servers on two identical desktop computers equipped with Intel i7-4790 processor, at 3.60 GHz over a 1Gbps LAN with an average round-trip ping of 0.3ms.

Our algorithm hand matches the one listed in Fig. 3, where the **Evaluate** step from Fig. 3 was implemented using the SPDZ software [7]. The preprocessing phase for this step was estimated using the LowGear protocol by Keller et al. [26], which is the fastest known protocol to produce triples for multiple parties with active security. We do not report on the timings for the feature extraction and reduction, since they can be done in the clear, locally by the external parties, which provide inputs to the MPC engine, and they are not privacy-sensitive.

EPIC – Simple Variant: We evaluated the computational performance, data sent over the network, and classification accuracy of EPIC on the default 2048 length feature from the output of Inception-v3. We report these experiment results in Table 2. Increasing the number of classes n (from 10, to 67, to 102) has a worsening effect on the performance, as the amount of data sent over the network scales linearly with n. The runtime of the online phase is affected less as n increases. Going from 10 classes (CIFAR-10) with 0.005 seconds runtime, to 102 classes (Caltech) with 0.03 seconds, is an increase factor of six, whereas for all other metrics it is roughly ten (i.e., linear in the number of classes).

In Table 3 we show that EPIC improves over Gazelle [25] in terms of every relevant metric on CIFAR-10: accuracy with 7%, total communication by 50x, and total runtime by 34x. This is because we start with secret shared (powerful) features obtained from public CNNs, whereas Gazelle [25] starts with an encrypted image. We expect Gazelle's timings to considerably improve, if they adopt our approach, starting from encrypted features produced by a public CNN.

EPIC – Complex Variant: To increase the performance of EPIC even further, we tried to minimize the feature size used, while still matching the classification accuracy achieved by Gazelle [25] or MiniONN [33] for CIFAR-10. In the end, we settled with $\mathsf{nc} = 128$, and then performed a grid search on γ for the MIT and Caltech datasets. Our results are reported in Table 4. Since the number of features decreases considerably from 2048 to 128 the timings decrease as well. For example, if we look at the online runtime compared to Gazelle [25], our solution improves by a factor of 700x and the total communication cost decreases by almost 500x. We do recognize that our setting is different from the one considered by Gazelle [25], but we see more the similarities, since the end goal is the same, namely to classify secret shared (or encrypted) images.

Our results indicate that general image recognition, and user's privacy can go well together. In fact we showed that securing the private classification comes

nearly for free. A stronger case for why CNN features with a Linear SVM should be considered, as a baseline benchmark is done by Razavian et al. [42].

Other optimizations: Note that one of the major improvements came from running the dot products on multiple threads, and doing the argmax operation in a tree-wise manner to decrease the number of communication rounds required.

Dataset	Runtime (s)			Communication (MB)			Accuracy
	Offline	Online	Total	Offline	Online	Total	%
CIFAR	0.36	0.005	0.37	24	0.33	24.33	88.8
MIT	2.43	0.02	2.45	161.94	2.24	164.18	72.2
Caltech	3.71	0.03	3.74	246.59	3.41	250	91.4

Table 2. 1 Gbps LAN timings for EPIC – Simple Variant on different datasets with a Linear SVM.

Framework	Runtime (s)			Communication (MB)			Accuracy
	Offline	Online	Total	Offline	Online	Total	%
MiniONN [33]	472	72	544	3046	6226	9272	81.61
Gazelle [25]	9.34	3.56	12.9	940	296	1236	81.61
EPIC	0.36	0.005	0.37	24	0.33	24.33	88.8

Table 3. 1 Gbps LAN timings for CIFAR-10 dataset on different frameworks. The EPIC – Simple Variant is compared to the state-of-the-art private classification solutions, and outperforms them in all metrics.

Dataset	Runtime (s)			Communication (MB)			Accuracy
	Offline	Online	Total	Offline	Online	Total	%
CIFAR	0.037	0.0003	0.037	2.472	0.027	2.5	81.74
MIT	0.259	0.002	0.261	17.22	0.180	17.4	64.4
Caltech	0.395	0.004	0.399	26.27	0.273	26.543	85.56

Table 4. 1 Gbps LAN timings for EPIC – Complex Variant on different datasets with a RBF-SVM and a 128 feature size.

Multiparty Setting. We benchmarked EPIC on different number of computers with the RBF-128 variant on the CIFAR-10 dataset and measured throughput (operations per second) for the online and offline phases in Fig. 4. For the

two party case our protocol can carry around 2650 evaluations per second. The throughput decreases with a growing number of parties and reaches 870 ops per second for the five parties case. Notice that the main bottleneck when executing these protocols is still the preprocessing phase, generating the necessary triples.

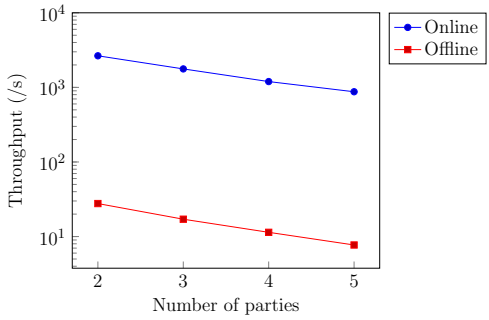


Fig. 4. Throughput of CIFAR-10 evaluations of secret features with RBF-128 EPIC for multiple parties.

Similar work. It is worth mentioning that we also implemented the method of Barnett et al. [4] in SPDZ, after fixing some security bugs such as cleartext PCA coefficients. They report 124s for one binary classification thus to extrapolate this to 10 classes takes roughly 1240s. To translate the work for Barnett et al. in SPDZ we used a feature extraction algorithm based on Histogram of Oriented Gradients (HOG) and then reduced their dimension using PCA. The reduced points where then plugged into a polynomial SVM to classify the inputs. This methodology yielded a 6.7s execution time of the online phase, and an expensive preprocessing phase of 12 hours for CIFAR-10. The classification accuracy was also poor (58%). This showed that the input dependent phase in MPC is faster than in FHE, by at least two orders of magnitude, confirming that our EPIC solution outperforms traditional attempts at classifying images using SVMs.

The closest work to ours that tried to solve the issue of linear SVM classification is a semi-honest 2-PC protocol due to Bost et al [6]. In this work party A owns the model and party B holds the features to be classified. To compare our method with theirs in an accurate manner we took their open sourced code and tailored it to our feature length (2048), input size (27 bits) and computational security $\lambda = 128$ and run it on our computers; whilst maintaining their statistical security of 100 bits. In Table 5 the method of Bost et al. [6] is benchmarked with the recent libraries (NTL-11.3.0, HElib, etc.). We then compare with EPIC using the same parameters as the ones used in the experiments of Bost et al.,

namely statistical security $\kappa = 100$ and computational security $\lambda = 128$, where the shares live in $\mathbb{F}_p$ and $p \approx 2^{128}$. For more details on the selection of the security parameters, we refer the reader to the full version of our paper [34]. EPIC has a faster online phase than Bost et al., by at least a factor of 20, at the cost of a slower preprocessing phase. This shows that the main bottleneck in the entire protocol is the triple generation, which deploys expensive cryptographic tools.

Method	Classes	Runtime (s)			Communication (MB)		
		Offline	Online	Total	Offline	Online	Total
[6]	10	0	0.48	0.48	0	5.36	5.36
EPIC	10	1.04	0.014	1.054	46.35	0.66	47.01
[6]	102	0	1.67	1.67	0	54.85	54.85
EPIC	102	10.72	0.083	10.8	475.96	6.68	482.64

Table 5. 1 Gbps LAN timings for EPIC – Simple Variant and Bost et al. with different number of classes.

5 Conclusion and Future Work

We have introduced EPIC, a private image classification system, trained with SVM, while having the input features extracted based on the techniques of transfer learning. We showed how to achieve privacy-preserving image classification in such a way that the task can be fully outsourced to a third, independent party. For our solution we deployed generic MPC tools and showed how to avoid the restricted two-party setting. Unlike all previous work, our approach provides active security, does not leak any information about the private images, nor about the classifier parameters, and it is orders of magnitude more efficient than the privacy-preserving classification solutions proposed in the literature.

Due to their highly accurate predictions, especially for multiclass classification tasks, CNNs have superseded SVM as the state-of-the-art for image classification. However, our work shows that in the privacy-preserving domain, SVM classification can still produce accurate results, as long as it is provided with high quality features. Thus, we chose to focus on improving the feature extraction phase, using a transfer learning, CNN-based approach, while avoiding the execution of such complex functions in the MPC domain. An interesting advantage of our solution is that it can be applied to the homomorphic encryption domain, since performing the linear operations has depth 1, and the costlier operation is computing the argmax, which requires to branch on secret comparisons.

Our experiments confirmed that there is a tradeoff between the complexity, and therefore also accuracy of the classification algorithms used, versus the efficiency of the privacy-preserving variants of the proposed solutions. In the active security model that we consider in this work, deploying CNNs in the same manner as they are used on cleartext data, is computationally prohibitive with current privacy-preserving methods.

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070. This work has been supported in part by the Research Council KU Leuven grants C14/18/067 and STG/17/019.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A System for Large-Scale Machine Learning. In: OSDI. pp. 265–283 (2016)
2. Aslett, L.J., Esperança, P.M., Holmes, C.C.: Encrypted Statistical Machine Learning: New Privacy Preserving Methods. arXiv:1508.06845 (2015)
3. Azizpour, H., Razavian, A.S., Sullivan, J., Maki, A., Carlsson, S.: Factors of Transferability for a Generic Convnet Representation. IEEE transactions on pattern analysis and machine intelligence **38**(9), 1790–1802 (2016)
4. Barnett, A., Santokhi, J., Simpson, M., Smart, N.P., Stainton-Bygrave, C., Vivek, S., Waller, A.: Image Classification using non-linear Support Vector Machines on Encrypted Data. IACR Cryptology ePrint Archive: 2017/857 (2017)
5. Barni, M., Orlandi, C., Piva, A.: A Privacy-Preserving Protocol for Neural-Network-Based Computation. In: Multimedia and security workshop. pp. 146–151. ACM (2006)
6. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine Learning Classification over Encrypted Data. In: Network and Distributed System Security Symposium (2015)
7. Bristol Crypto: SPDZ-2: Multiparty computation with SPDZ, MASCOT, and Overdrive offline phases. https://github.com/bristolcrypto/SPDZ-2 (2018)
8. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-Preserving Classification on Deep Neural Network. IACR Cryptology ePrint Archive: 2017/35 (2017)
9. Chase, M., Gilad-Bachrach, R., Laine, K., Lauter, K., Rindal, P.: Private Collaborative Neural Network Learning. IACR Cryptology ePrint Archive: 2017/762 (2017)

10. Dahl, M.: Private Image Analysis with MPC: Training CNNs on Sensitive Data using SPDZ. https://mortendahl.github.io/2017/09/19/private-image-analysis-with-mpc/ (2018)
11. Damgård, I., Damgård, K., Nielsen, K., Nordholt, P.S., Toft, T.: Confidential Benchmarking based on Multiparty Computation. IACR Cryptology ePrint Archive: 2015/1006 (2015)
12. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical Covertly Secure MPC for Dishonest Majority–or: Breaking the SPDZ Limits. In: ESORICS. pp. 1–18. Springer (2013)
13. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption. In: CRYPTO, pp. 643–662. Springer (2012)
14. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A Large-Scale Hierarchical Image Database. In: CVPR. pp. 248–255. IEEE (2009)
15. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In: ICML. pp. 647–655 (2014)
16. Drineas, P., Mahoney, M.W.: On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. Journal of Machine Learning Research **6**(Dec), 2153–2175 (2005)
17. Fei-Fei, L., Fergus, R., Perona, P.: Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. In: CVPR. pp. 178–178. IEEE (2004)
18. Gastaldi, X.: Shake-Shake Regularization. arXiv:1705.07485 (2017)
19. Gentry, C.: Fully Homomorphic Encryption using Ideal Lattices. In: STOC. pp. 169–178 (2009)
20. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In: ICML. pp. 201–210 (2016)
21. Graepel, T., Lauter, K., Naehrig, M.: ML Confidential: Machine Learning on Encrypted Data. In: ICISC. pp. 1–21. Springer (2012)
22. He, K., Zhang, X., Ren, S., Sun, J.: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual recognition. In: ECCV. pp. 346–361. Springer (2014)
23. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely Connected Convolutional Networks. In: CVPR. pp. 4700–4708. IEEE (2017)
24. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding. In: ACMMM. pp. 675–678. ACM (2014)
25. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In: USENIX. pp. 1651–1668 (2018)
26. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: EUROCRYPT. pp. 158–189. Springer (2018)

27. Khan, F.S., van de Weijer, J., Anwer, R.M., Bagdanov, A.D., Felsberg, M., Laaksonen, J.: Scale Coding Bag of Deep Features for Human Attribute and Action Recognition. Machine Vision and Applications **29**(1), 55–71 (2018)

28. Krizhevsky, A., Hinton, G.: Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto (2009)

29. Laur, S., Lipmaa, H., Mielikäinen, T.: Cryptographically Private Support Vector Machines. In: SIGKDD. pp. 618–624. ACM (2006)

30. Lin, K.P., Chen, M.S.: Privacy-Preserving Outsourcing Support Vector Machines with Random Transformation. In: SIGKDD. pp. 363–372. ACM (2010)

31. Lin, K.P., Chen, M.S.: On the Design and Analysis of the Privacy-Preserving SVM Classifier. Knowledge and Data Engineering **23**(11), 1704–1717 (2011)

32. Lindell, Y., Pinkas, B.: Privacy Preserving Data Mining. In: CRYPTO. pp. 36–54. Springer (2000)

33. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious Neural Network Predictions via MiniONN Transformations. In: SIGSAC. pp. 619–631. ACM (2017)

34. Makri, E., Rotaru, D., Smart, N.P., Vercauteren, F.: EPIC: Efficient Private Image Classification (or: Learning from the Masters). IACR Cryptology ePrint Archive: 2017/1190 (2017)

35. Mohassel, P., Zhang, Y.: SecureML: A System for Scalable Privacy-Preserving Machine Learning. In: S&P. pp. 19–38. IEEE (2017)

36. Orlandi, C., Piva, A., Barni, M.: Oblivious Neural Network Computing via Homomorphic Encryption. EURASIP Journal on Information Security p. 18 (2007)

37. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

38. Quattoni, A., Torralba, A.: Recognizing Indoor Scenes. In: CVPR. pp. 413–420. IEEE (2009)

39. Rahimi, A., Recht, B.: Random Features for Large-Scale Kernel Machines. In: NIPS. pp. 1177–1184 (2008)

40. Rahimi, A., Recht, B.: Weighted Sums of Random Kitchen Sinks: Replacing Minimization with Randomization in Learning. In: NIPS. pp. 1313–1320 (2009)

41. Rahulamathavan, Y., Phan, R.C.W., Veluru, S., Cumanan, K., Rajarajan, M.: Privacy-Preserving Multi-Class Support Vector Machine for Outsourcing the Data Classification in Cloud. Dependable and Secure Computing **11**(5), 467–479 (2014)

42. Razavian, A.S., Azizpour, H., Sullivan, J., Carlsson, S.: CNN Features off-the-shelf: An Astounding Baseline for Recognition. In: CVPRW. pp. 512–519. IEEE (2014)

43. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In: ASIACCS. pp. 707–721. ACM (2018)

44. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On Data Banks and Privacy Homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978)

45. Rouhani, B.D., Riazi, M.S., Koushanfar, F.: DeepSecure: Scalable Provably-Secure Deep Learning. In: ACM/ESDA/DAC. pp. 1–6. IEEE (2018)

46. Shokri, R., Shmatikov, V.: Privacy-Preserving Deep learning. In: SIGSAC. pp. 1310–1321. ACM (2015)

47. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the Inception Architecture for Computer Vision. In: CVPR. pp. 2818–2826. IEEE (2016)

48. Teo, S.G., Han, S., Lee, V.C.: Privacy Preserving Support Vector Machine using non-linear Kernels on Hadoop Mahout. In: CSE. pp. 941–948. IEEE (2013)

49. Thrun, S.: Is Learning the n-th Thing any Easier than Learning the First? In: NIPS. pp. 640–646 (1996)

50. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing Machine Learning Models via Prediction APIs. In: USENIX. pp. 601–618 (2016)

51. Vaidya, J., Yu, H., Jiang, X.: Privacy-Preserving SVM Classification. Knowledge and Information Systems **14**(2), 161–178 (2008)

52. Vapnik, V.N.: Statistical Learning Theory, vol. 3. Wiley New York (1998)

53. Williams, C.K., Seeger, M.: Using the Nyström Method to Speed up Kernel Machines. In: NIPS. pp. 682–688 (2001)

54. Yang, T., Li, Y.F., Mahdavi, M., Jin, R., Zhou, Z.H.: Nyström Method vs Random Fourier Features: A Theoretical and Empirical Comparison. In: NIPS. pp. 476–484 (2012)

55. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How Transferable are Features in Deep Neural Networks? In: NIPS. pp. 3320–3328 (2014)

Chapter 9

Rabbit: Efficient Comparison for Secure Multi-Party Computation

Publication data

E. Makri, D. Rotaru, F. Vercauteren, S. Wagh. "Rabbit: Efficient Comparison for Secure Multi-Party Computation" In *International Conference on Financial Cryptography and Data Security.* Springer, 2021, *To appear.*

Rabbit: Efficient Comparison for Secure Multi-Party Computation

Eleftheria Makri[1,5], Dragos Rotaru[2,1], Frederik Vercauteren[1], and Sameer Wagh[3,4]

[1] imec-COSIC, KU Leuven, Belgium
[2] Cape Privacy
[3] University of California, Berkeley, USA
[4] Princeton University, Princeton, USA
[5] ABRR, Saxion University of Applied Sciences, The Netherlands
emakri@esat.kuleuven.be, dragos@capeprivacy.com,
frederik.vercauteren@kuleuven.be, swagh@berkeley.edu

Abstract. Secure comparison has been a fundamental challenge in privacy-preserving computation, since its inception as Yao's millionaires' problem (FOCS 1982). In this work, we present a novel construction for general n-party private comparison, secure against an active adversary, in the dishonest majority setting. For the case of comparisons over fields, our protocol is more efficient than the best prior work (edaBits: Crypto 2020), with $\sim 1.5\times$ better throughput in most adversarial settings, over $2.3\times$ better throughput in particular in the passive, honest majority setting, and lower communication. Our comparisons crucially eliminate the need for bounded inputs as well as the need for statistical security that prior works require. An important consequence of removing this "slack" (a gap between the bit-length of the input and the MPC representation) is that multi-party computation (MPC) protocols can be run in a field of smaller size, reducing the overhead incurred by privacy-preserving computations. We achieve this novel construction using the commutative nature of addition over rings and fields. This makes the protocol both simple to implement and highly efficient and we provide an implementation in MP-SPDZ (CCS 2020).

Keywords: Secure Comparison · Multi-party Computation · Unconditional Security · Dishonest Majority.

1 Introduction

After years of active research, both in theoretical results and system building, multi-party computation (MPC) is becoming practical as a paradigm. Recent research results and practical implementations [13,1], deployment of MPC in real-life applications [3], as well as organizations beyond academia offering commercial MPC solutions [30,27,26], confirm that MPC is reaching maturity. However, MPC, just like any other cryptographic primitive deployed to enhance privacy, comes at a significant efficiency penalty, in terms of computation and communication. While some research

focuses on tailoring MPC solutions to a particular problem, to compensate for this efficiency penalty, other works focus on improving the efficiency of fundamental MPC building blocks, which are applicable to a wide variety of problems.

Secure comparison is an important problem in multi-party computation – it involves the comparison of two or more secret values in a privacy-preserving manner. Comparison is a fundamental building block, necessary for the realization of various larger tasks: from online auctions to big data analytics and machine learning. Given the privacy considerations that today's digital infrastructure entails, protocols for secure comparison are a fundamental MPC tool in privacy-preserving applications.

Since the introduction of the secure comparison problem by Andrew Yao in 1982 [34] as the millionaires' problem, research efforts have pushed the frontiers of performance of this primitive. MPC has traditionally been efficient either on linear operations, when it is based on arithmetic circuits, or on non-linear operations, when it is based on Boolean circuits. Recent applications require a combination of linear and non-linear operations, and they are most of the time addressed with solutions based on arithmetic circuits, because these are significantly more efficient than Boolean circuits for the linear part, which presents itself as the bulk of the computation. Given the non-linear nature of the comparison operation, protocols for secure comparison still remain a bottleneck for privacy-preserving computation. Thus, any improvement in this line of work has a compounding impact on improving the overall efficiency of privacy-preserving computations.

In this work, we present a novel comparison protocol that is secure against an active adversary in the dishonest majority setting and holds for general n-party computation. Our work improves upon the state-of-the-art protocol for comparison in dishonest majority in both the total time and communication by a factor of two for the OT-based preprocessing. In addition, our protocol is easy to implement requiring no heavy cryptography. Notably, our protocol is highly conducive to amortization and preprocessing, which makes it attractive for deployment in real-life applications, as these are important considerations in building practical secure systems.

1.1 Our Contribution

We present Rabbit[1], a novel secure comparison protocol, which leverages the commutative nature of addition over rings and fields. Our protocol exploits recent advances in the generation and deployment of *doubly authenticated shared bits* (daBits [25]), which are bits living both in $\mathbb{F}_p$ and in $\mathbb{F}_{2^k}$, as well as *extended doubly authenticated bits* (edaBits [14]), which correspond to shared integers in the arithmetic domain, whose bit decomposition is shared in the binary domain. The proposed comparison is more efficient than previously proposed secure comparison protocols, while at the

[1] The name is an extension of the daBit [25], maBit [24] and edaBit [14] line of work.

same time removing the dependence on bounds and statistical parameters. This allows the MPC engines used for our secure comparison to be smaller than the ones required by previous protocols, which has a positive impact on the concrete efficiency of the MPC protocols. Concretely we make the following contributions:

(i) **Novel comparison protocol:** We propose Rabbit, a novel secure comparison protocol based on the commutative nature of addition over rings and fields. Rabbit is a general n-party protocol and crucially eliminates the need for any "slack" – a statistically larger dataspace to ensure security of computations, and thus enables computations over smaller datatypes. For instance, to compute over 64-bits, prior works require the use of 128-bit datatypes, while we can support these computations in standard 64-bit datatypes.

(ii) **Security:** Since we eliminate the slack and keep an exact tab of overflows, our comparison algorithms are unconditionally secure. However, our protocols use edaBits [14], which requires us to account for a statistical security parameter. Lastly, when implemented in a larger body of MPC computation, our protocols inherit the security properties of the platform. The current implementation of our protocols uses MP-SPDZ [13], which further reduces the overall security to computational, against active adversaries in the dishonest majority setting.

(iii) **Simplicity and Efficiency:** Our protocol is straightforward to implement. As shown in Fig. 1b, it is merely a few lines of code in MP-SPDZ. This also makes our protocol highly amenable to secure implementation. As for efficiency, the benefits of our work over the state-of-the-art are most pronounced in the case of comparison over fields. In this case, we improve end-to-end computations such as secure evaluation of ResNet-50 up to 2x faster, albeit at a higher communication.

1.2 Technical Overview

Our central focus in this work is to propose novel and efficient protocols for secure comparison. Comparison protocols usually rely on statistical security or bit-decomposition combined with prefix computation to achieve the results. We observe that:

(i) When considering arithmetic secret shares, the bit encoding modulus overflow of secrets enables exact integer relations between the secret, the secret shares, and the modulus.

(ii) Using the commutativity of addition over standard structures, such as rings and fields, we can express a sum in two different ways and thus equate the corresponding constraint equations.

These two observations together enable more efficient protocols for comparisons. More specifically, the core idea behind our comparison protocols lies in our ability to detect when a sum over a particular modulus overflows (i.e., wraps around), and when this happens we can correct it. Observe that given two integers $x,y \in \mathbb{Z}_M$, their sum $x+y \bmod M$ is less than either of the two summands, iff the sum wrapped around the modulus. That is, given a comparison function:

$$\mathsf{LT}(\cdot,\cdot):\mathbb{Z}\times\mathbb{Z}\rightarrow\{0,1\} \subseteq \mathbb{Z} : \begin{cases} \mathsf{LT}(x,y)=1 & \text{if } (x<y); \\ \mathsf{LT}(x,y)=0 & \text{otherwise,} \end{cases}$$

we can compute the modular sum $x+y \bmod M$, by performing computations over the integers as:

$$x+y \bmod M = x+y-M\cdot\mathsf{LT}(x+y \bmod M,x) = x+y-M\cdot\mathsf{LT}(x+y \bmod M,y)$$

This is due to the observation that $\mathsf{LT}(x+y \bmod M,x)$ (or $\mathsf{LT}(x+y \bmod M,y)$) is true, iff the sum wrapped around. Given that the $\mathsf{LT}(\cdot,\cdot)$ function detects (i.e., outputs true) when a wrap around happens, we can indeed realize the modular sum, while performing computations over the integers, by conditionally subtracting the quantity of the wrap around (i.e., M), when $\mathsf{LT}(\cdot,\cdot)$ returns true.

Notation. We use $[x]_N$ to denote the sharing of a secret x in the ring $\mathbb{Z}_N$. We primarily consider two values of the modulus: $N=M$ and $N=2$, where M is a fixed constant, set to either a prime p, or a power of two 2^k. The types of sharings are:

(i) $[x]_M$, where the secret is $x \in \mathbb{Z}_M$ or $[b]_2$, where the secret is a bit $b \in \mathbb{F}_2$;

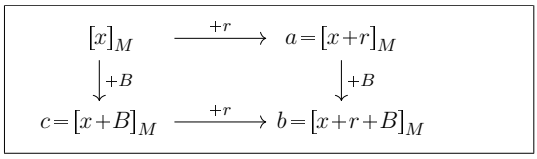
(ii) $[x]_M$ and $[x_0]_2,\ldots[x_{m-1}]_2$ such that $x=\sum_{i=0}^{m-1} x_j \cdot 2^j \pmod M$ and $M<2^m$ (this is also known as an edaBit [14])

Similarly, given a (public) constant value $R \in \mathbb{Z}_M$, we denote by $R_0,\ldots,R_{m-1}$ the bit decomposition of R, and by R_i its individual bits (at the corresponding position i).

2 Comparison Protocols

In this section we present our comparison protocols and their workings on a step-by-step basis. Then, for each presented protocol, we also show correctness. We do not provide any formal proofs of security of our protocols, as these follow trivially from the arithmetic black box functionality paradigm [11]. We present the protocols in the following order:

(i) First, we present the protocol Π_{LTBits} (Fig. 3), which realizes a comparison between a secret bit-decomposed value, and a public value, and outputs a secret bit indicating the result of the comparison. This is a building block that uses prefix computation for comparison.



(a) Intuition behind Rabbit comparison protocol

```
# Secure comparison in MP-SPDZ
k = program.bit_length
r, r_bits = sint.get_edabit(k, True)
a = (x+r).reveal()
b = a + M - R

w1 = LTBits(a, r_bits, k)
w2 = LTBits(b, r_bits, k)
w3 = (b < M - R)

movs(s, sint.conv(w1-w2+w3))
return
```

(b) Rabbit code snippet

Fig. 1: Our protocol relies on the commutative properties of addition over rings/fields as shown in Fig. 1a. This diagram indicates the two different ways we can obtain the value b. The $[\cdot]_M$ notation indicates that the corresponding values or sums are taken modulo M. The horizontal arrows indicate addition of a uniformly random value $r \in \{0,...,M-1\}$, used to mask the secret input of the comparison x (so that we can later open it without information leakage, to perform a comparison). The vertical arrows indicate addition of a known constant $B \in \{0,...,M-1\}$ related to the public quantity to be compared against. These two ways of computing the sum b, are necessary for the comparison protocol between a secret value x and a public constant $M-B$. The code on the right (Fig. 1b) shows the simplicity of implementing our protocol, implemented in this case in the MP-SPDZ codebase [13].

(ii) Second, we introduce the protocol Π_{LTC} (Fig. 4), which invokes Π_{LTBits} and performs a comparison between a secret value (without bit-decomposition), and a public value, where the output is a secret bit indicating the result of the comparison.

(iii) Third, we present a specialized comparison protocol, Π_{ReLU} (Fig. 5), that can be applied when the modulus is a power of 2 and the public constant against which we compare is half the modulus. Note that this is an important case, as it corresponds to computation of the ReLU function, which is widely used in machine learning.

(iv) Finally, in Π_{LTS} (Fig. 6), we show how to generalize Π_{LTC} to compare two secret shared values, where once again the output is a secret bit.

Note that given our novel approach of comparison, there is a difference between secret-public constant comparison (Π_{LTC}) and secret-secret comparison (Π_{LTS}), which often comes for free when using standard techniques that require a slack. For more details on this, we refer the reader to Section 4. Finally, for all proposed protocols, the output can either be an element of $\mathbb{Z}_M$ or $\mathbb{F}_2$ (depending on the needs of the follow-up computations) indicating the result of the comparison. An overview of all our comparison protocols, their inputs, and their interdependencies is given in Fig. 2.

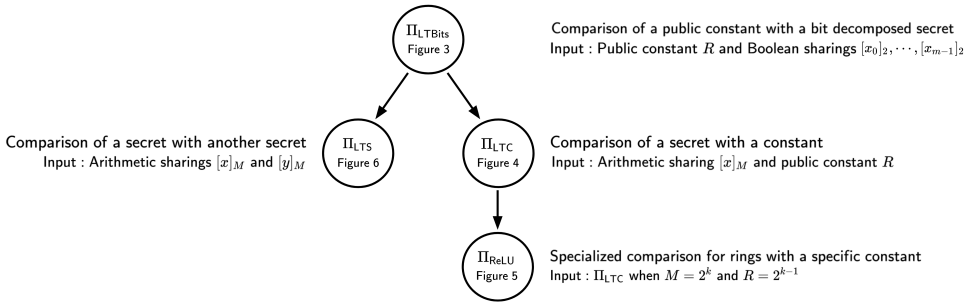


Fig. 2: Proposed comparison protocols, their inputs, and their interdependencies.

2.1 Comparison with Bitwise Shared Input – LTBits Protocol

The protocol Π_{LTBits}, listed in Fig. 3, follows a standard bit decomposition idea to privately compute a secret bit, indicating the result of a comparison. It is essentially an adaptation of the BIT-LT protocol by Damgård *et. al.* [9], which instead of two secret bit-decomposed inputs (that BIT-LT receives), it receives one bitwise secret shared input and a public arithmetic value to compare upon, while its output is a secret Boolean value indicating the result of the comparison. Notably, each component of our bit-decomposed secret lives in $\mathbb{F}_2$, unlike Damgård *et. al.*'s [9] construction, where each secret bit lives in $\mathbb{F}_p$. The protocol LTBits computes the following:

1. The XOR of each bit of the secret input $[x_i]_2$ value with the corresponding bit of the public value R_i. This results in a bit-string $[y_0]_2$,...$[y_{m-1}]_2$ with ones on all positions where the bits of the values to be compared differ.

2. A prefix OR (circuit computes for each position i of a bit vector, the OR between all previous bits in the vector up to position i. - more details in Catrina and de Hoogh [6]) of the previously computed bits $[y_i]_2$, which results in a vector $[z_i]_2$ of 0's followed by 1's with the transition from 0 to 1 occurring at the first bit where the secret and the public value differ.

3. In this step, the previous vector is converted into a vector $[w_i]_2$, $i = 0,...,m-1$ of all 0's and a single 1 at the index of the first differing bit.

4. In the last step, we take the inner product between the vector $\mathbf{w}$ (which is a vector of 0's in all positions, except for the position of the first differing bit of the values to be compared) and the bits of the public value R. This inner product results in 0, if at the position of the differing bit R was 0, which further implies that x is larger than R, and it results in 1 otherwise. We have computed the value $[(x < R)]_2$, but we are actually after $[(R < x)]_2$, thus $1 - [(x < R)]_2$ concludes the protocol.

Less Than Bits $\Pi_{\mathsf{LTBits}}(R, [x_0]_2, \ldots [x_{m-1}]_2)$

Inputs: Secret value x shared bitwise, such that parties hold $[x_0]_2, \ldots [x_{m-1}]_2$, where $x = \sum_{i=0}^{m-1} x_j \cdot 2^j$, and public value R.

Outputs: Compute the Boolean value $[c]_2 = [(R \leqslant x)]_2$.

Protocol: Complete steps 1-3 for all $i \in \{0,1,\ldots,m-1\}$

1. Parties compute $[y_i]_2 = [x_i]_2 \oplus R_i$.
2. Parties compute $[z_i]_2 = \vee_{j=i}^{m-1} [y_j]_2$ using PrefixOR circuit.
3. Parties compute $[w_i]_2 = [z_i]_2 - [z_{i+1}]_2$, where $z_m = 0$.
4. Output $[c]_2 = 1 - [(x < R)]_2$, where $[(x < R)]_2 = \sum_{i=0}^{m-1} R_i \cdot [w_i]_2$.

Fig. 3: Protocol for comparison between an input shared bitwise and a public value.

Correctness of Π_{LTBits}: To see the correctness of Π_{LTBits}, note the following series of observations:

1. To compare two numbers, we start from the most significant bit (MSB) and look for the first bit where the two numbers differ. This is precisely what is computed in Step 1 of Π_{LTBits}. Thus, $y_{m-1}, \ldots, y_0$ contains a series of 0's, followed by a 1, which in turn is followed by bits that are irrelevant to the comparison.

2. As a consequence, $z_{m-1}, \ldots, z_0$ contains a series of 0's followed by 1's starting at the first location where x_i and R_i differ. Let $k \in \{0,\ldots,m-1\}$ be the largest index where $x_i \neq R_i$. Thus, $w_i = 1$ iff $i = k$ and $w_i = 0$ otherwise.

3. Finally, multiplying w_i by R_i ensures the following:

$$\mathsf{output} = \begin{cases} 1 & \text{if } R_k = 1,\ x_k = 0 \text{ (implying } R > x) \\ 0 & \text{otherwise (implying } R \leqslant x) \end{cases}$$

■

2.2 Comparison with a Constant – LTC Protocol

The protocol Π_{LTC}, listed in Fig. 4, is a comparison protocol between a shared secret value, and a public constant. Unlike Π_{LTBits}, it does not require the secret input value to be bitwise secret shared, but it invokes the protocol Π_{LTBits} twice. These two invocations can be parallelized, decreasing the total number of rounds of the comparison protocol. Π_{LTC} requires an edaBit as an input. An edaBit is a shared value in the arithmetic domain, accompanied by its own bit decomposition in the binary domain [14]. The core idea behind this comparison protocol is that addition in a ring or field is commutatitve as explained in Fig. 1a.
The Π_{LTC} protocol proceeds as follows:

Less Than Constant $\Pi_{\mathsf{LTC}}([x]_M, R)$

Inputs: Value x secret shared, such that parties hold $[x]_M$, a shared $\mathsf{edaBit}([r]_M, [r_0]_2, \ldots, [r_{m-1}]_2)$ and public value R.

Outputs: Compute the Boolean value $[(x < R)]_2$.

Protocol:

1. Parties compute the value $[a]_M = [x + r]_M$ (and $[b]_M = [x + r + M - R]_M$).
2. Parties open the value a ($b \equiv a + M - R$ can be opened locally).
3. Parties compute the following quantities:
   - $[w_1]_2 = \Pi_{\mathsf{LTBits}}(a, [r_0]_2, \ldots, [r_{m-1}]_2)$.
   - $[w_2]_2 = \Pi_{\mathsf{LTBits}}(b, [r_0]_2, \ldots, [r_{m-1}]_2)$.
   - $w_3 = (b < M - R)$.
4. Output $[w]_2 = 1 - ([w_1]_2 - [w_2]_2 + w_3)$ or use one classical daBit to output $[w]_M$.

Fig. 4: Protocol for comparison between *an input shared in* $\mathbb{Z}_M$ *and a public value* R for any modulus M (in particular, M can be 2^k or a prime p).

1. Using the arithmetic value $[r]_M$ of the random edaBit from the input, the parties mask the input value x, computing $[a]$.
2. $[a]$ is opened, without revealing any information about x.
3. The parties then do the following:
   (a) Invoke Π_{LTBits} to compare the masked value $[a]$ against the random edaBit (in bitwise sharing), resulting in $[w_1]_2$.
   (b) Invoke Π_{LTBits} to compare $b = [a + M - R]_M$ against the random edaBit (in bitwise sharing), resulting in $[w_2]_2$.
   (c) Compare in the clear b against the public value $B = M - R$, resulting in w_3.
4. Finally, they conclude the comparison test by computing $[w]_2 = 1 - ([w_1]_2 - [w_2]_2 + w_3)$. This equation follows from the way we exploited the commutative property of addition, and its correctness is explained in the next paragraph. The output at this step is the binary value indicating the result of the comparison, shared in $\mathbb{F}_2$. Depending on the follow-up computations in the larger MPC protocol that uses the comparison, if the next input needs to be arithmetic, a classical daBit [25] can be used to transform the representation of this bit in $\mathbb{Z}_M$.

Correctness of Π_{LTC}**:** Let us denote by $[x]$ the value of $x \in \mathbb{Z}_M$, i.e., the modular reduction in $\{0, 1, \ldots, M-1\}$. We are interested in securely computing the Boolean

value ($x < R$), for R a public constant. Furthermore, let $\mathsf{LT}(x,y)$ be defined as follows:

$$\mathsf{LT}(x,y) = \begin{cases} 1 & \text{if } x < y \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Recall from Section 1.2 that the $\mathsf{LT}(x,y)$ function enables writing exact integer relations for the sum of two numbers as follows:

$$\begin{aligned} [x+y] &= [x] + [y] - M \cdot \mathsf{LT}([x+y],[x]) \\ &= [x] + [y] - M \cdot \mathsf{LT}([x+y],[y]) \end{aligned} \tag{2}$$

To be consistent with the notation followed in Fig. 1a, we define $B = M - R$, and $c = [x+B]$. We then use the commutative nature of addition to represent the sum $b = [x+r+B]$ in two different ways, as shown in Fig. 1a. Using Eq. 2 for the two additions in the top path and noting that $a,b,B \in \mathbb{Z}_M$:

$$\begin{aligned} b = [a+B] &= a + B - M \cdot \mathsf{LT}(b,B) \\ &= x + r - M \cdot \mathsf{LT}(a,r) + B - M \cdot \mathsf{LT}(b,B) \end{aligned} \tag{3}$$

Similarly, using Eq. 2 for the two additions on the bottom path , we get:

$$\begin{aligned} b = [c+r] &= c + r - M \cdot \mathsf{LT}(b,r) \\ &= x + B - M \cdot \mathsf{LT}(c,B) + r - M \cdot \mathsf{LT}(b,r) \end{aligned} \tag{4}$$

Equating the RHS of Eq. 3, and Eq. 4, we get:

$$\mathsf{LT}(a,r) + \mathsf{LT}(b,B) = \mathsf{LT}(c,B) + \mathsf{LT}(b,r) \tag{5}$$

Recall that the result we are after is $\mathsf{LT}(x,R)$, which is equivalent to $(1 - \mathsf{LT}(c,B))$, since $B = M - R$, and $c = [x+B]$. Thus, from Eq. 5 we have $\mathsf{LT}(c,B) = 1 - (\mathsf{LT}(a,r) + \mathsf{LT}(b,B) - \mathsf{LT}(b,r))$, which is exactly what we compute in Step 4 of Π_{LTC}. Finally, to complete the proof, we reiterate that $\mathsf{LT}(c,B) = 0$ iff $(x < R)$ and that $\mathsf{LT}(\cdot,\cdot)$ correctly computes the function defined by Eq. 1. ■

The correctness argument above holds, under the assumption that the edaBit functionality produces correct and secure tuples. We note that the security of our Π_{LTC} protocol reduces to statistical, due to the use of edaBits (which offer statistical security). Since edaBits are sampled on a bit-by-bit basis, and then composed (first locally, by each party, and then globally) to the arithmetic sharing, we require both for correctness, and for statistical security that our modulus M is close to a power of 2. We further elaborate on this matter in Section 4.3.

2.3 Π_{ReLU} – Special Case of Π_{LTC} for $R = 2^{k-1}$, $M = 2^k$

Π_{LTC} is a general comparison protocol for comparing against *any* public value. However, a special case of interest is when the modulus is a power of 2 and the public constant to be compared against is half the modulus. When considering privacy-preserving alternatives to machine learning, the use of fixed-point arithmetic converts the widely used $\mathsf{ReLU}(x) = \max(x,0)$ function to the above comparison, when considering such a special modulus (power of 2). In this case, where $R = 2^{k-1}$ and $M = 2^k$, the protocol can be optimized further to improve performance. We present this optimized protocol in Fig. 5. This comparison setting is useful in a number of privacy-preserving machine learning frameworks [22,32], where fixed point encoding transforms the ReLU function into a comparison with $R = 2^{k-1}$ and $M = 2^k$. In this case, we can simplify our protocol to open the masked value $a = [x+r]$ (Step 1 of the protocol), subtract the mask r from it using a binary circuit in the secret shared domain (Steps 2, 3, 4 of the protocol), and extract the MSB of this result (Step 6). This way we are essentially extracting the MSB of x. This replaces the overhead of two invocations of Π_{LTBits} with a single invocation of a binary addition protocol (Π_{BitAdder}). The computation in Step 4 can also be used to perform comparisons when $R = 2^\ell$ is another power of two, however that would require additional computation over the bits $s_{k-1},\ldots,s_\ell$.

$\mathsf{ReLU}\ \Pi_{\mathsf{ReLU}}([x]_{2^k}, 2^{k-1})$

Inputs: Value x secret shared, such that parties hold $[x]_{2^k}$, a shared edaBit $([r]_{2^k}, [r_0]_2, \ldots, [r_{k-1}]_2)$ and the public value 2^{k-1}.

Outputs: Compute shares $[y]_{2^k}$ where $y = x$ if $(x \leqslant 2^{k-1})$ and 0 otherwise.

Protocol:

1. Parties compute the value $[a]_{2^k} = [x+r]_{2^k}$ and open a.
2. Parties locally compute $[t_0]_2, \ldots [t_{k-1}]_2 = [1-r_0]_2, \ldots, [1-r_{k-1}]_2$
3. Parties set $a_0, \ldots a_{k-1}$ to be the bits of $(a+1)$.
4. $[s_0]_2, \ldots, [s_{k-1}]_2, [s_k]_2 = \Pi_{\mathsf{BitAdder}}(a_0, \ldots, a_{k-1}, [t_0]_2, \ldots, [t_{k-1}]_2)$[a].
5. Output $[s_{k-1}]_2$ or use one classical daBit to output $[s_{k-1}]_{2^k}$ if only the derivative of ReLU is required in the computation.
6. Use one multiplication triple and output $y = [x]_{2^k} \cdot [s_{k-1}]_{2^k}$.

[a] Π_{BitAdder} is a circuit performing addition over bitwise shared values.

Fig. 5: Protocol for comparison between *an input shared in* $\mathbb{Z}_{2^k}$ and 2^{k-1}.

Correctness of Π_{ReLU}: Observe that in this special case comparison with the constant 2^{k-1} where the modulus is 2^k, the MSB of the secret input defines the result of

the comparison. Our protocol essentially performs a bit decomposition of the input $[x]_{2^k}$ by masking it (using the arithmetic version of the edaBit) and then again subtracting this mask in a binary circuit (using the binary version of the edaBit). This results in the bit decomposition of x, and by extracting its MSB we conclude the comparison, and hence the computation of this ReLU function.

Remark – Optimizing Π_{ReLU}: Note that Step 4 in Figure 5 can be optimized as we only require a single bit $[s_{k-1}]_2$. In particular, this requires $\log_2 k$ rounds and $k\log_2 k$ invocations of bit-triples. This can be reduced to $\log_2 k$ rounds and $2k-2$ bit-triples by simply modifying the MSB values and using a prefix computation protocol Π_{PreOpL} (cf [6]). We modify the most significant bit of the input tuple to be (1,0) before passing to the Π_{PreOpL}. Consequently, the second element of the output tuple of the Π_{PreOpL} protocol is the carry bit $[s_{k-2}]_2$ and thus $[s_{k-1}]_2$ can be computed locally as the XOR of the MSB's of the two bits and the bit $[s_{k-2}]_2$.

2.4 Comparison with Secret – LTS Protocol

While the protocol described in Sec. 2.2 provides an efficient way to compare with a public constant, the protocol described in this section, Π_{LTS}, listed in Fig. 6, enables the comparison of two secret values x and y. In most prior works, due to the use of a slack or bounds on inputs, the corresponding protocols for these two settings are nearly identical. In our case, the elimination of slack requires slightly different protocols. We provide a brief discussion on applications of either of these protocols in Sec. 4.2.

Each step of the protocol Π_{LTS} computes the following:

1. Parties mask the input values $[y]$ and $[x]$ using the arithmetic shares of two random edaBits $[r]$ and $[r']$, resulting in shared values $[b]$ and $[a] \in \mathbb{Z}_M$.
2. These masked values are opened (without revealing any information about x or y) and the value $T \equiv a+b \pmod M$ is computed locally.
3. The parties then perform the following computations:
   (a) Using Π_{LTBits}, a secret comparison between the open value b and the bitwise sharing of the edaBit r, and store the result $[w_1]_2$.
   (b) A similar comparison between a and the bitwise sharing of r', and store the output in $[w_2]_2$.
   (c) Check in the clear whether $(T < b)$, and store this value in w_3.
   (d) Compute a circuit for bitwise addition of two binary (secret) vectors, where the result is a bitwise secret shared vector of the bits of $(r+r')$.
   (e) Extract the last carry bit from the binary adder (Step 3d) as $[w_4]_2$.

Less Than Secret $\Pi_{\mathsf{LTS}}([x]_M,[y]_M)$

Inputs: Values x and y secret shared, such that parties hold $[x]_M$ and $[y]_M$, two shared $\mathsf{edaBits}([r]_M,[r_0]_2,...,[r_{m-1}]_2)$ and $([r']_M,[r'_0]_2,...,[r'_{m-1}]_2)$.

Outputs: Compute the Boolean value $[(x<y)]_2$.

Protocol:

1. Parties compute the values $[b]_M=[y+r]_M$, $[a]_M=[r'-x]_M$
2. Parties open the values a and b, and compute $T\equiv a+b \pmod M$ locally.
3. Parties compute the following quantities:
   - $[w_1]_2=\Pi_{\mathsf{LTBits}}(b,[r_0]_2,...,[r_{m-1}]_2)$.
   - $[w_2]_2=\Pi_{\mathsf{LTBits}}(a,[r'_0]_2,...,[r'_{m-1}]_2)$.
   - $w_3=(T<b)$.
   - $[s_0]_2,...,[s_{m-1}]_2,[s_m]_2=\Pi_{\mathsf{BitAdder}}([r_0]_2,...,[r_{m-1}]_2,[r'_0]_2,...,[r'_{m-1}]_2)$.
   - $[w_4]_2=[s_m]_2$
   - $[w_5]_2=\Pi_{\mathsf{LTBits}}(T,[s_0]_2,...,[s_{m-1}]_2)$.
4. Output $[w]_2=[w_1]_2+[w_2]_2+w_3-[w_4]_2-[w_5]_2$, or use one classical daBit and output $[w]_M$.

Fig. 6: Protocol for comparison between *two arithmetic inputs shared in* $\mathbb{Z}_M$, for any modulus M (in particular, M can be 2^k or a prime p).

(f) Finally, using Π_{LTBits}, compare the value T against the bitwise secret sharing of $r+r'$ (computed in Step 3d), and store the output in $[w_5]_2$.

4. In the end, the parties conclude the comparison protocol by computing the output $[w]_2=[w_1]_2+[w_2]_2+w_3-[w_4]_2-[w_5]_2$. This final step, similarly to the LTC protocol follows from the way we exploit the commutative nature of addition, and we show correctness subsequently. The final output is the binary sharing of the comparison result, which can be transformed to a shared bit in $\mathbb{Z}_M$ if needed.

Correctness of Π_{LTS}**:** Following the same notation set-up as in Sec. 2.2 for Π_{LTC}, we denote by $[x]$ the value of $x\in\mathbb{Z}_M$, and the function $\mathsf{LT}(x,y)$ as defined in Eq. 1. We are interested in securely computing the Boolean value $(x<y)$, for x and y two secret shared values in $\mathbb{Z}_M$. The intuition for our protocol is presented in Fig. 7 and follows the same idea as in Π_{LTC}, viz., computing a sum in two different ways and using Eq. 2 to find a constraint between the various wrappings around the modulus.

First note that $[x]<[y]$ iff $\mathsf{LT}([y-x],[y])=1$. We then mask the inputs y and $-x$ using the two $\mathsf{edaBits}$: $[b]=[y+r]$, $[a]=[r'-x]$. Finally, we look at computing the value $[T]=[y-x+r+r']$ in two different ways, as the sum of a and b, and as the

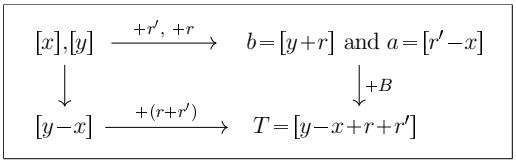


Fig. 7: Intuition behind the comparison protocol for two secret values, once again based on the commutative nature of addition over rings and fields.

sum of $y-x$ and $r+r'$. Looking at the addition using the first way, we first open the values a and b, and write the exact integer relation (using Eq. 2):

$$T = b + a - M \cdot \mathsf{LT}(T,b) \tag{6}$$

We can also write similar expressions for b and a,

$$\begin{aligned} b &= [y] + [r] - M \cdot \mathsf{LT}(b,[r]) \\ a &= [-x] + [r'] - M \cdot \mathsf{LT}(a,[r']) \end{aligned} \tag{7}$$

Thus the first expression for the sum T is given by (combining Eqs. 6, 7):

$$T = [y] + [r] - M \cdot \mathsf{LT}(b,[r]) + [-x] + [r'] - M \cdot \mathsf{LT}(a,[r']) - M \cdot \mathsf{LT}(T,b) \tag{8}$$

Grouping the terms differently and computing the sum using the latter expression:

$$T = [y-x] + [r+r'] - M \cdot \mathsf{LT}(T,[r+r']) \tag{9}$$

Once again, $[y-x]$ and $[r+r']$ can be expanded using Eq. 2 as:

$$\begin{aligned} [y-x] &= [y] + [-x] - M \cdot \mathsf{LT}([y-x],[y]) \\ [r+r'] &= [r] + [r'] - M \cdot \mathsf{LT}([r+r'],[r]). \end{aligned} \tag{10}$$

Plugging Eq. 10 into Eq. 9, and equating that with the expression in Eq. 8, we get the following expression for $\mathsf{LT}([y-x],[y])$, the quantity of interest:

$$\mathsf{LT}([y-x],[y]) = \mathsf{LT}(b,[r]) + \mathsf{LT}(a,[r']) + \mathsf{LT}(T,b) - \mathsf{LT}([r+r'],[r]) - \mathsf{LT}(T,[r+r'])$$

This completes the correctness proof. To generate an efficient protocol for this expression, the final observation is that $\mathsf{LT}([r+r'],[r])$ is generated as a by-product of the computation required to generate the bit decomposition of $r+r'$ from the bit decompositions of r,r' (to enable a call to Π_{LTBits}). ■

3 Evaluation

We implement our protocol in the MP-SPDZ Framework [13]. The entire protocol is a handful of lines of python code, as shown in Fig. 1b, and reads directly from the pseudocode; this makes it highly amenable to implementation. We evaluate our protocol over various MPC settings and a brief summary of our experiments is provided below:

(i) **Throughput of Comparisons:** In this experiment, we measure the throughput of comparison operations and compare this with prior art. These results are presented in Sec. 3.1.

(ii) **Private Evaluation of ResNet-50:** We provide benchmarks for evaluating ResNet-50 [17] using dishonest majority privacy-preserving computation. We use the state-of-the-art matrix triple generation algorithm [7] and combine that with our comparison protocol and compare that against the prior art [7,14]. These results are presented in Sec. 3.2.

Set-up Details: We use an MPC set-up similar to prior works [14,25,24]. Each party is run on an Intel(R) Core(TM) i9-9900 CPU @ 3.10GHz with 128GB of RAM over a 10Gb/s network switch with an average round-trip ping time of 1ms. For the WAN setting we use two or three machines depending on the protocol wich are equipped with Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz and 54GB of RAM while the network capability was slowed down using the Linux **tc** command limiting the bandwidth to 100Mb/s and 100ms round-trip ping time.

3.1 Throughput of Rabbit comparisons

We conduct experiments in all combinations of the possible adversarial models (active, passive), adversarial settings (honest majority, dishonest majority), and domains (OT-based in $\mathbb{Z}_{2^k}$, OT-based in $\mathbb{F}_p$, HE-based in $\mathbb{F}_p$), and in both the LAN and WAN network settings. Table 1 provides a summary of the primitives used as preprocessing (i.e., offline cost) for a Rabbit comparison, vs. an edaBit comparison [14], their online round complexity, security, and the need for slack, in $\mathbb{Z}_{2^k}$ and in $\mathbb{F}_p$. As in Escudero *et. al.* [14], we benchmark the time required for a million comparisons between two (DM) or three (HM) servers described in the setup above. Table 2, 3 show the number of comparisons per second (throughput) and communication per party (kbits) for a single operation in the LAN and WAN settings respectively. Our protocol improves prior art in runtime and communication by upto $2\times$, and in all cases, achieves these without any slack.

Communication for Π_{LTC} over $\mathbb{F}_p$. Note that our protocol incurs higher communication cost, when performing comparisons over fields. This is due to the use of

Sub-protocols	Rabbit		edaBits Comp. [14]	
	$\mathbb{Z}_{2^k}$	$\mathbb{F}_p$	$\mathbb{Z}_{2^k}$	$\mathbb{F}_p$
edaBits	1:$\{k\}$	1:$\{k\}$	1:$\{l\}$	2:$\{l-m+s,m\}$
daBits	1	1	1	1
ANDs	$3(k-1)$	$k\log_2 k^*$	$3(l-1)$	$2(k-1)$
# Rounds	$2+\log_2 k$	$\log_2 k$	$2\log_2 l$	$2\log_2 k$
Security, slack	Perfect, No	Statistical, No	Statistical, Yes	Statistical, Yes

Table 1: Theoretical complexity comparison of exact comparison functionality over $\mathbb{Z}_{2^k}$ and $\mathbb{F}_p$ where k is the bit-size of the datatypes, l is the $\log_2$ bound on the inputs/data, and m refers to the number of bits to be truncated.

a more expensive Prefix OR computation. Prior works encode the data in a larger dataspace and simply extract the MSB for the comparison. In a manner similar to the optimization from Π_{LTC} to Π_{ReLU}, we can extract the MSB to compute a comparison. This operation requires using a prefix computation protocol Π_{PreOpL} (cf [6]), which has a linear overhead of $2(k-1)$ bit-triples in $\log_2 k$ rounds – matching that of edaBits [14]. If a different encoding is used, where positive and negative numbers are determined by comparison with $\lfloor p/2 \rfloor$, the same protocol can be used with statistical correctness, determined by the specific choice of prime (with a small gap between p and 2^k). A suitable choice of prime p would also further lower the prepossessing time, when performed using HE.

3.2 Neural Network Evaluation

In this section, we provide benchmarks for using our approach for comparison on evaluating the ResNet-50 architecture [17]. In our experiments, we consider neural network inference over 64-bit datatypes and compare the offline and online performance of our protocol with the state-of-the-art protocols with active security in the dishonest majority setting. For prior art, we use the recent protocol for matrix triple generation [7] in conjunction with our Π_{LTC} comparison protocol. The results are summarized below.

The work of Chen *et. al.* [7] requires the plaintext modulus to be 128-bits, due to the slack required in the comparison. In this work, we eliminate that slack and hence only require generation of matrix triples using homomorphic encryption (HE) with a plaintext space of 64-bits. While Chen *et. al.* [7] require a 128-bit modulus and $N = 2^{15}$ (degree of the cyclotomic polynomial), we can generate 64-bit triples. This enables us to run the algorithm with lower HE parameters (and consequently bet-

		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	Comm.(kb)	Thru.(ops/s)	Comm.(kb)
Dishonest Majority	**Active**	2^k (OT)	2936	1252.4	3038	1252.2
		p (OT)	1537	2847.0	1056	4458.6
		p (HE)	1495	1678	1495	1635.99
	Passive	2^k (OT)	165368	39.5	172211	38.3
		p (OT)	73947	87.8	51478	132.2
		p (HE)	65750	67.63	41175	41.71
Honest Majority	**Active**	2^k	117607	5.62	116616	5.54
		p	88780	9.43	41028	19.62
	Passive	2^k	5706569	0.5	5600265	0.5
		p	1421412	0.96	472316	1.58

Table 2: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over LAN* for 16 threads, with 2 million comparisons in total.

ter performance). We use $N = 2^{14}$, a plaintext modulus of 64-bits and a ciphertext modulus of 480. With a conservative analysis this leaves enough room for 40-bits of statistical security. We set the block size to 64 instead of 128 and thus pack 4 matrices in a single ciphertext (compared to 2 in Chen *et. al.* [7]). We list the sizes of matrices required for the computations in ResNet-50 and then measure the time required (and communication overhead) for matrix triple generations using these different set-ups. We run the protocols on a similar set-up as Chen *et. al.* [7], using a 5Gb/s LAN bandwidth and about 300 Mbps WAN bandwidth. Hence, just for the triple generation, our communication complexity reduces by about 60% and the total time by about 40% of [7] for the same set of triple generations (LAN and WAN settings are fairly similar as the protocols are compute dominated). Furthermore, our computational burden for the matrix triple computations reduces from about 72GB to 9.3GB – a critical improvement for systems based on HE.

We also run the offline and online computations for the comparisons in ResNet-50 and compare the total time. Our protocol takes about 11 hours and 2883.3 GB of communication. When compared to prior art of Chen *et. al.* [7], they evaluate the same network in about 24hrs with 2036 GB (using improved comparisons using edaBits). Thus, our work is $2\times$ faster albeit uses slightly more communication due to the communication gap for Rabbit and edaBit for dishonest majority within a characteristic p field. Thus, our comparison protocol, combined with the improvement in the triple generation phase due to slack elimination, provides a significant throughput improvement over state-of-the-art MPC protocols for neural network evaluation.

		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	Comm.(kb)	Thru.(ops/s)	Comm.(kb)
Dishonest Majority	**Active**	2^k (OT)	33	1237	33	1237
		p (OT)	1.37	29646	0.37	112594
		p (HE)	2	19089	N/A	N/A
	Passive	2^k (OT)	596	39.26	604	38.18
		p (OT)	366	87.59	245	131
		p (HE)	427	67.01	431	41.71
Honest Majority	**Active**	2^k	5444	5.54	5488	5.52
		p	1639	16.96	1463	19.53
	Passive	2^k	15096	0.49	15182	0.49
		p	11492	0.96	7640	1.53

Table 3: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over WAN*. All numbers were produced using 2 million comparisons with 8 threads, except in the active security, dishonest majority field cases where we used only 32,000 comparisons due to time constraints. Note that for the active security, dishonest majority field case with HE preprocessing, the 54GB RAM machines ran out of memory due to the large ciphertexts kept in memory by MP-SPDZ - for Rabbit there were no memory issues as the memory footprint is reduced to half due to ciphertexts that only need to accommodate 64-bits plaintexts.

4 Discussion

In this section, we provide a deeper discussion on the following aspects of this work. We (1) elaborate on our central contribution of removing the slack and how it enables computation over smaller data types; (2) we discuss applications of these protocols; and (3) provide an analysis of the statistical security provided by our protocol along with the choice of modulus for the case of fields.

4.1 Elimination of "Slack" in Comparisons

One important contribution of this work is the elimination of a "slack" between the inputs (in other words the computable part of the data) and the actual size of the datatypes used in the MPC engines. Note that prior work in the dishonest majority setting requires a slack to accommodate for the statistical parameter. Commonly, this statistical parameter, which is necessary to ensure security, is at least 40-bits. This implies that the actual datatypes used in the MPC are at least 40-bits longer than the values we need to compute upon. As a consequence, prior work requires 128-bit datatypes for the MPC, necessary to support 64-bit computations. On the contrary, our comparison protocol achieves exact comparison without the need for any slack and thus operates on smaller, 64-bit datatypes. As shown in Section 3.2, when the

slack removal is combined with recent advances such as the contributions of the work of Chen *et. al.* [7], the smaller MPC datatypes enable faster triple generation, reduce the communication and computational overhead and increase the overall efficiency of the MPC computations, beyond secure comparisons.

4.2 Applications to Machine Learning and Beyond

Privacy-preserving machine learning, which is of increasing interest in the field of MPC, often relies on efficient protocols for computing ReLU, a non-linear function that is given by $\mathsf{ReLU}(x) = \max(x,0)$. Using fixed-point encoding, computation of the ReLU function reduces to a comparison with an encoding of 0 (i.e., a constant). Given that this non-linear function is the bottleneck of many state-of-the-art secure machine learning protocols [21,18], our proposed protocol improves this entire line of work.

The thresholding operation is yet another application where we require a comparison with a public constant. In image processing and computer vision, threasholding is used for segmenting images (e.g., turn a grayscale image into a binary one). In particular, it replaces a pixel with a black (resp. white) pixel, if the image intensity is less (resp. greater) than a fixed constant. In yet another application, Cryptography for #metoo [19], the system heavily relies on the use of public value thresholding. In adversarial machine learning, algorithms for robustness that work over privacy-preserving computation also require thresholdings with small public values. In all these applications, the functionality can be efficiently achieved using our comparison with constant protocol. Thus, our efficient comparison with constant protocol, Π_{LTC} (Sec. 2.2), is deployable on several application scenarios.

On the other hand, there are applications, where secure comparisons with a constant do not suffice, but a comparison between two values that are both secret is required. In such cases, our comparison with secret protocol, Π_{LTS} (Sec. 2.4) can be deployed. Applications in this line of work go as far in the past as the first instance of the problem: Yao's millionaires' problem [34], and include amongst others also secure auctions [4], and secure linear programming [28].

4.3 Statistical Security

We remark that Π_{LTBits}, Π_{LTC}, Π_{ReLU} and Π_{LTS} are all inherently information theoretically secure. However, the current implementation needs to account for a small statistical security due to the use of $\mathsf{edaBits}$ [14]. The protocol for edaBit generation produces shares:

$$[r]_M \text{ and } \{[r_i]_2\}_{i=0}^{m-1} \text{ such that } r \equiv \sum_{i=0}^{m-1} r_i \cdot 2^i \pmod{M} \tag{11}$$

In particular, for the correctness of Π_{LTC} in Sec. 2.2, we require that $r = \sum r_i \cdot 2^i$, and this condition is different from Eq. 11 in a subtle yet important way. In the case

where $M = 2^m$, this does not raise an issue. However, in all other cases, in particular including the field case, we have $2^{m-1} < M < 2^m$, and so we can have $r = (\sum r_i \cdot 2^i) - M$. In this case, the correctness of Π_{LTC} does not hold, as the set of sharings $\{[r_i]_2\}_{i=0}^{m-1}$ does not correspond to the bit decomposition of r. To address this issue, we note that this failure probability depends on the size of the gap between the modulus and the bounding power of 2 in relation to the modulus. The failure probability is given by:

$$\mathsf{Failure\ probability} = \frac{2^m - M}{2^m} \tag{12}$$

which is simply the probability that r is between M and 2^m. Thus, if $\delta = 2^m - M$, the failure probability can be made small for suitable choice of $\delta/2^m$. Thus, in practice, we choose the largest 64-bit prime $p = 2^{64} - 59$ for our implementation. This gives our protocol a failure probability of less than 2^{-59}. However, from a security point of view, for statistical hiding, we use the fact that $r \xleftarrow{R} \{0,1,\cdots,2^m - 1\}$ when reduced modulo M is still close to uniform in $\mathbb{Z}_M$ (to ensure the masked value is hidden). If the former distribution is D_1 and the latter is D_2, then this statistical distance can be computed exactly as given in Eq. 13. Thus, the statistical closeness can also be made negligible by a suitable choice of $\delta/2^m$. A union bound over the two expressions (Eq. 12 and 13) allows us to achieve both correctness and privacy with a statistical parameter close to 58-bits.

$$\begin{aligned} \mathsf{Statistical\ closeness} &= \mathsf{Distance}(D_1, D_2) \\ &= \frac{1}{2}\left[\left(\sum_{i=0}^{\delta-1} \frac{2}{2^m} - \frac{1}{M}\right) + \left(\sum_{i=\delta}^{2^m-1} \frac{1}{M} - \frac{1}{2^m}\right)\right] \\ &= \frac{\delta \cdot (M - \delta)}{M \cdot 2^m} \leqslant \frac{\delta}{2^m} \end{aligned} \tag{13}$$

Furthermore, we note that one can use rejection sampling as follows: run Π_{LTBits} over the bit decomposition of r and the modulus M to check if $r \geqslant M$. If this is the case then reject the sample. This way we can eliminate such edaBits and note that the rejections happen with probability similar to the expression in Eq. 12 and is thus ideal once again when the prime p is close to a power of 2.

As an aside, the closer the prime is to the power of two, the lower is the failure probability. However, when combining with other protocols, such as those mentioned in Sec. 3.2, there are other considerations in choosing the prime. For instance, for efficiency reasons BFV [15,5] requires special prime modulus, where $p-1$ has a large factor (around 2^{14} - 2^{16}). One such prime is $p = 2^{64} - 83$, where $33196 \mid p - 1$ and $\phi(33196) = 16128$ (with ϕ the Euler's Totient function), which would be secure given the 16k degree and appropriately chosen modulus q.

Lastly, we reiterate that when combined with a larger MPC platform, the overall security is set by the weaker between the MPC platform and the protocol or algorithm, and hence when using implementations such as SPDZ [12], BDOZa [2], SPD$\mathbb{Z}2^k$ [10], as the underlying MPC platform, our security reduces to computational.

5 Comparison with Related Work

After the seminal work of Yao [34], which operates in the two-party setting, and is based on garbled circuits, many works studied the problem of secure comparisons, both in the two-party [8,29,35], as well as in the multi-party setting [9,23,6,20]. In this work, we focus on the general n-party setting. Damgård *et. al.* [9] were the first to tackle the challenge of secure, constant-round bit decomposition of secret shared inputs, which is a necessary building block for most comparison protocols. In the same work [9], they extend and apply their bit-decomposition protocol to develop a secure comparison protocol (amongst other applications). Their comparison protocol works in the general n-party setting, with any underlying linear secret sharing scheme (LSSS), and provides unconditional security against active adversaries (assuming that the multiplication protocol of the LSSS is also actively secure), in the honest majority setting.

Improving upon the complexity of Damgård *et. al.*'s [9] bit decomposition, comparison, equality, and interval test protocols, Nishide and Ohta [23] provide new, simplified protocols. In addition, Nishide and Ohta [23] construct new secure comparison, equality, and interval test protocols, which do not rely on bit decomposition. For their deterministic equality test protocol that is independent of bit decomposition, Nishide and Ohta [23] apply a masking technique similar to the one we use in our comparison protocol: they use a random shared value that the parties possess both in its $\mathbb{F}_p$ and in its bit decomposed form to mask and afterwards open the secret shared input of the equality test.

In an attempt to design comparison protocols with concrete efficiency instead of asymptotic, Catrina and de Hoogh [6] propose several versions of secure equality and comparison tests. Their protocols run in logarithmic number of rounds, in the bit-length of the values to be compared, but also with logarithmic communication cost (instead of the usually linear communication cost). The efficiency of these protocols comes also at the cost of statistical, instead of unconditional security and have been adopted and implemented in a number of MPC platforms (e.g., [13,1]). Our comparison protocol, in combination with the recent advances in the generation of daBits [25], and edaBits [14] performs concretely better than the one of Catrina and de Hoogh [6], while offering unconditional (instead of statistical) security in $\mathbb{Z}_{2^k}$.

Lipmaa and Toft [20] propose three different comparison protocols. Only one of these comparison protocols works for the general n-party setting with active security,

Protocol	Communication		Computation		Rounds	Security	Adversary	Setting
	Offline	Online	Offline	Online				
[9]	-	$\mathcal{O}(\ell\log\ell)$	-	$\mathcal{O}(\ell\log\ell)$	$\mathcal{O}(1)$	perfect	active	HM
[23]	-	$\mathcal{O}(\ell)$	-	$\mathcal{O}(\ell)$	$\mathcal{O}(1)$	perfect	passive	HM
[6]	-	$\mathcal{O}(\log\ell)$	-	$\mathcal{O}(\log\ell)$	$\mathcal{O}(\log\ell)$	statistical	passive	HM
[20]	$\mathcal{O}(\ell)$	$\mathcal{O}(\log\ell)$	$\mathcal{O}(\log\ell)$	$\mathcal{O}(\log\ell)$	$\mathcal{O}(\log\ell)$	statistical	active	HM
Rabbit	$\mathcal{O}(\ell)$	$\mathcal{O}(\ell\log\ell)$	$\mathcal{O}(\ell)$	$\mathcal{O}(\ell)$	$\mathcal{O}(\log\ell)$	perfect*	active	DM

Table 4: Comparison of the related work in the n-party setting in terms of offline, and online communication and computation complexity; in terms of rounds; in terms of security; and in terms of adversarial model and adversarial settings supported. In the context of adversarial setting HM stands for honest majority, while DM stands for dishonest majority. *perfect security holds only when the underlying secret sharing scheme operates over $\mathbb{Z}_{2^k}$.

and while it offers sublinear online communication complexity, it is not constant-round and it has linear offline communication cost. Like other protocols in the literature [29,8], the core of [20] lies in the idea of splitting the two strings to be compared into smaller, equal length blocks, and perform the comparison on the first block where they differ. This way the problem of comparison only needs to be addressed on smaller strings (the blocks), and equality testing can be applied to the larger strings (to allow for the necessary reduction of the size of the blocks on which comparison is to be performed). Other recent concretely-efficient comparison protocols such as [16,32,33,31] also eliminate the need for a slack but operate in fixed adversarial models and are tied to a 3-party MPC setting.

In Table 4 we detail the asymptotic costs and security features of the related work in secure comparisons for the general n-party setting. It is important to remark that most prior secure comparison protocols require the values to be compared to be smaller than the space where the comparison takes place. Although this may result in efficient protocols for the particular comparison operations, it also requires a larger MPC engine to perform all (other) computations. Essentially, this means that all adjacent computations should be performed in a larger space, and all values to be communicated throughout the protocol need to be larger by a factor proportional to the necessary slack for the secure comparison. Our protocol crucially overcomes this limitation.

6 Conclusion

In this work, we propose novel comparison protocols for general n-party computation. Our protocols enjoy perfect security, when we operate over $\mathbb{Z}_{2^k}$, and crucially

eliminate the need for "slack" – a larger dataspace to compute secure comparisons, enabling computations over smaller datatypes. In terms of concrete efficiency, our protocols improve prior art by twice for most adversary structures, while keeping a smaller communication complexity. Given that comparisons are a fundamental secure computation primitive, many MPC applications can benefit from our protocols.

Acknowledgements

This work was supported in part by the Research Council KU Leuven grant C14/18/067, and by CyberSecurity Research Flanders with reference number VR20192203, and by ERC Advanced Grant ERC-2015-AdG-IMPaCT.

References

1. Abdelrahaman Aly, Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Nigel P. Smart, and Tim Wood. SCALE-MAMBA v1.2: Documentation, 2018.
2. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic Encryption and Multiparty Computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–188. Springer, 2011.
3. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure Multiparty Computation Goes Live. In *International Conference on Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.
4. Peter Bogetoft, Ivan Damgård, Thomas Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. A Practical Implementation of Secure Auctions based on Multiparty Integer Computation. In *International Conference on Financial Cryptography and Data Security*, pages 142–147. Springer, 2006.
5. Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology—CRYPTO*, pages 868–886. Springer, 2012.
6. Octavian Catrina and Sebastiaan de Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer, 2010.
7. Hao Chen, Miran Kim, Ilya Razenshteyn, Dragoş Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning. In *Advances in Cryptology—ASIACRYPT*, 2020.
8. Geoffroy Couteau. New Protocols for Secure Equality Test and Comparison. In *Applied Cryptography and Network Security (ACNS)*, 2018.
9. Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally Secure Constant-Rounds Multi-Party Computation for Equality, Comparison, Bits and Exponentiation. In *Theory of Cryptography Conference (TCC)*, pages 285–304. Springer, 2006.

10. 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 *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

11. Ivan Damgård and Jesper Buus Nielsen. Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In *Annual International Cryptology Conference*, pages 247–264. Springer, 2003.

12. Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

13. Data61. MP-SPDZ: Versatile Framework for Multi-party Computation, 2019. `https://github.com/data61/MP-SPDZ`.

14. Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. Cryptology ePrint Archive, Report 2020/338, 2020. `https://eprint.iacr.org/2020/338`.

15. Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

16. Wataru Fujii, Keiichi Iwamura, and Masaki Inamura. Secure Comparison and Interval Test Protocols Based on Three-Party MPC. In *6th International Conference on Information Systems Security and Privacy, ICISSP 2020*, pages 698–704. SciTePress, 2020.

17. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

18. Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, 2018.

19. Benjamin Kuykendall, Hugo Krawczyk, and Tal Rabin. Cryptography for# metoo. In *Privacy Enhancing Technologies Symposium (PETS)*, 2019.

20. Helger Lipmaa and Tomas Toft. Secure Equality and Greater-Than Tests with Sublinear Online Complexity. In *International Colloquium on Automata, Languages, and Programming*, pages 645–656. Springer, 2013.

21. Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.

22. Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE Symposium on Security and Privacy (S&P)*, 2017.

23. Takashi Nishide and Kazuo Ohta. Multiparty Computation for Interval, Equality, and Comparison without Bit-Decomposition Protocol. In *International Workshop on Public Key Cryptography*, pages 343–360. Springer, 2007.

24. Dragos Rotaru, Nigel P Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively Secure Setup for SPDZ. Cryptology ePrint Archive, Report 2019/1300, 2019. https://eprint.iacr.org/2019/1300.

25. Dragos Rotaru and Tim Wood. Marbled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security. In *International Conference on Cryptology in India*, pages 227–249. Springer, 2019.

26. Sepior. https://sepior.com/, 2020.

27. Sharemind. https://sharemind.cyber.ee/, 2020.

28. Tomas Toft. Solving Linear Programs Using Multiparty Computation. In *International Conference on Financial Cryptography and Data Security*, pages 90–107. Springer, 2009.

29. Tomas Toft. Sub-Linear, Secure Comparison with Two Non-Colluding Parties. In *International Workshop on Public Key Cryptography*, pages 174–191. Springer, 2011.

30. Unbound. https://www.unboundtech.com/, 2020.

31. Sameer Wagh. *New Directions in Efficient Privacy Preserving Machine Learning*. PhD thesis, Princeton University, 2020.

32. Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-Party Secure Computation for Neural Network Training. In *Privacy Enhancing Technologies Symposium (PETS)*, 2019.

33. Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. In *Privacy Enhancing Technologies Symposium (PETS)*, 2021.

34. Andrew C Yao. Protocols for Secure Computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

35. Ching-Hua Yu and Bo-Yin Yang. Probabilistically Correct Secure Arithmetic Computation for Modular Conversion, Zero Test, Comparison, MOD and Exponentiation. In *International Conference on Security and Cryptography for Networks*, pages 426–444. Springer, 2012.

Chapter 10

Full-Threshold Actively-Secure Multiparty Arithmetic Circuit Garbling

Publication data

E. Makri, T. Wood. "Full-Threshold Actively-Secure Multiparty Arithmetic Circuit Garbling". In *IACR Cryptology ePrint Archive*, 2019, 1098.

Full-Threshold Actively-Secure Multiparty Arithmetic Circuit Garbling

Eleftheria Makri[1,2] Tim Wood[1,3]

[1] imec-COSIC, KU Leuven, Belgium.
[2] ABRR, Saxion University of Applied Sciences, The Netherlands.
[3] University of Bristol, UK.

Abstract. In this work, we show how to garble arithmetic circuits with full active security in the general multiparty setting, secure in the full-threshold setting (that is, when only one party is assumed honest). Our solution allows interfacing Boolean garbled circuits with arithmetic garbled circuits. Previous works in the arithmetic circuit domain focused on the two-party setting, or on semi-honest security and assuming an honest majority – notably, the work of Ben-Efraim (Asiacrypt 2018) in the semi-honest, honest majority security model, which we adapt and extend. As an additional contribution, we improve on Ben-Efraim's selector gate. A selector gate is a gate that given two arithmetic inputs and one binary input, outputs one of the arithmetic inputs, based on the value of the selection bit input. Our new construction for the selector gate reduces the communication cost to almost half of that of Ben-Efraim's gate. This result applies both to the semi-honest and to the active security model.

Keywords: Arithmetic Garbling · Active Security · Efficient Selector Gate

1 Introduction

Garbled circuits have been an indispensable cryptographic tool in the field of secure computation since the seminal work of Yao [26]. From a theoretical point of view, garbled circuits are important as they provide the means by which we can construct *constant-round* secure computation protocols, originally only in the two-party setting, but later generalised to the multiparty setting, following the paradigm of Beaver et al. [5]. In the two-party setting, garbled circuits are typically *Boolean* circuits executed between two asymmetric parties – a garbler and an evaluator. However, many secure computation problems require arithmetic operations to emulate integer arithmetic, which are inefficient to realise with a Boolean circuit (e.g., requiring 1000 AND gates for an addition mod p and 100000 AND gates for a multiplication mod p, for $p \approx 2^{128}$). Towards the

goal of efficient constant-round computation of arithmetic circuits, one theoretical approach was given by Applebaum et al. [2] and more recently a practical solution was proposed by Ball et al. [4], in the two-party setting.

In this work we focus on *multiparty* arithmetic garbling. The work of Ben-Efraim [6] was the first to explore multiparty garbling of arithmetic circuits, and gave protocols secure in the presence of a passive adversary in the honest-majority setting. The goal of multiparty arithmetic garbling protocols is the functionality $\mathcal{F}_{\mathsf{AC}}$ for computing an arithmetic circuit, given in Figure 1. This functionality is the goal of all Multiparty Computations (MPC), but offers only security *with abort* instead of full *robustness*, in which honest parties can always obtain the correct output after the initial inputs are provided, or *fairness*, in which honest parties always receive the output if the corrupt parties receive it.

Functionality $\mathcal{F}_{\mathsf{AC}}$

Let $\mathcal{S}$ denote the ideal-world adversary and $A \subsetneq [n]$ the indexing set of corrupt parties.

Evaluate On input $(\mathsf{Evaluate}, C, \mathbf{x}^i)$ from each party P_i, or $\mathcal{S}$ if $i \in A$, if C is an arithmetic circuit over $\mathbb{F}_p$ and $\mathbf{x}^i \in \mathbb{F}_p^{t_i}$ for all i such that $\sum_{i=1}^{n} t_i$ is the arity of C, send $y := C(\mathbf{x}^1, \ldots, \mathbf{x}^n)$ to $\mathcal{S}$ and await a message Abort or OK from $\mathcal{S}$. If the message is OK, then send y to all honest parties and halt; otherwise, send the message Abort to all honest parties and halt.

Fig. 1: Functionality $\mathcal{F}_{\mathsf{AC}}$ for evaluating an arithmetic circuit, secure *with abort*.

In our approach, we allow a (limited) combination of arithmetic and Boolean circuits, as this appears to be desirable for many real-world applications. From the simplest motivating example that one can consider, such as the one of conditional summation that Ben-Efraim [6] suggests, to the most complicated computations, such as evaluation of Machine Learning (ML) algorithms, a combination of arithmetic with Boolean gates is required to yield an efficient solution. Machine Learning as a Service is becoming increasingly popular, and when privacy concerns arise, secure computation solutions should be deployed. The most commonly used ML algorithms (e.g., Support Vector Machines (SVMs) and Neural Networks) contain one or more components that require linear operations – for which arithmetic operations are more appropriate – and one or more components that require non-linear operations, such as argmax or sign computation – where Boolean computation is best. Thus it seems sensible to attempt to support both types of gates to achieve efficient solutions to realistic applications.

Related Work. Our work combines the work of Ben-Efraim [6], and Ball et al. [4], and extends them in such a way as to achieve full-threshold active secu-

rity by using recent actively-secure secret-sharing-based MPC to construct the circuit, a technique initiated by Lindell et al. [19]. In the work of Ball et al., which is based on some of the techniques discussed also in the work of Malkin et al. [20], the authors propose a two-party arithmetic garbling scheme, secure in the presence of a semi-honest adversary, where the arithmetic takes place in a ring isomorphic to a cyclic group of primorial modulus. They show how to use a Chinese Remainder Theorem (CRT) representation of the inputs (and intermediate values) of the circuit to achieve great performance gains over the straightforward conversion of ring elements to binary. In this approach, garbling of linear gates (e.g., addition and scalar multiplication) requires no communication and can be viewed as an arithmetic analogue of the FreeXOR technique due to Kolesnikov and Schneider [18] for Boolean circuits; multiplication, exponentiation by (public) constant, and high fan-in gates are also significantly improved beyond the naïve implementations. However, operations such as comparison of two numbers remain challenging, and prohibitively costly in the CRT representation. To overcome this issue, Ball et al. suggested a method to convert CRT numbers to a positional number system other than the binary system, namely the primorial mixed radix (PMR) system. Although highly improved over the straightforward (convert to binary) approach, the solution is still costly.

The work of Ben-Efraim [6] is secure in the presence of a passive adversary and assumes an honest majority, and involves a circuit construction comprising a mixture of arithmetic and Boolean gates. Ben-Efraim's construction also allows linear operations to be performed for free, while for multiplication gates a "designated" solution is proposed, inspired by the half-gates approach of Zahur et al. [27], extended to the multiparty setting. This is because projection gates (that is, gates that convert values in one ring to the equivalent values in another ring) are difficult to achieve in the multiparty setting, unlike the two-party setting, where as shown by Ball et al. [4], general projection gates are feasible.

Unfortunately, row-reduction techniques [21, 23] in the Boolean setting, and also applied in [4], cannot be directly applied in the multiparty setting as protocols for more than two parties are (usually) symmetrical – that is, every party acts both as garbler and evaluator. However, by elegantly re-applying a variation of the half-gates approach [27], Ben-Efraim proposes a construction for a "designated" selector gate solution (i.e., a gate which selects one out of two arithmetic inputs u and v, based on a third, binary input b) that reduces computation cost. Specifically, after describing the construction of a straightforward selector (projecting the bit to characteristic p, and then performing a multiplication using the standard multiplexing equation $u + (v - u)b$), Ben-Efraim demonstrates the designated selector gate, which has the same communication cost as the straight-

forward one ($2p + 2$ ciphertexts), but it improves the computation cost by 33% (i.e., 2 decryptions for the designated construction, instead of 3).

Concurrently and independently of our work, Ball et al. [3] propose a series of optimisations over the previous state-of-the-art in the two-party setting [4], which is tailored to the garbling of neural networks. One of their main technical contributions is the new mixed-modulus half-gate, which allows efficiently multiplying circuit wires from different domains. This can be thought of as a generalisation of the alternative selector gate that we present in this work, as we can only multiply bit wires by arithmetic wires, while their construction is not limited to bits. While our method can only treat mixed-modulus half-gate multiplications if one of the two domains is the $\mathbb{F}_2$, the approach of Ball et al. [3] is generalisable to multiplication of wires from any (different) domain. This is achieved by exploiting the asymmetry between the parties in the two-party case, where we can choose certain labels to only be used in one of the parties' half-gates. This does not extend to the multiparty garbling setting, which is our focus, because all parties play the role of the garbler. Still, we maintain that garbled multiplication of an integer by a bit is indeed the most commonly occurring mixed-modulus multiplication (e.g., selector gates). Note that the communication cost of our approach is almost the same as the cost of the approach of Ball et al. [3] (in the case of multiplying by a bit). The second contribution of that work is an improved mixed-radix addition, which is important for increasing the efficiency of the non-linear parts of a garbled neural network. Mixed-radix operations (other than the ones where the one operand is base 2) do not appear to extend readily to the multiparty case.

Our Contribution. We continue the study of Ben-Efraim [6] of multiparty garbling of circuits that contain both arithmetic and Boolean gates. Ben-Efraim [6] showed how to construct a designated selector gate in this setting, based on an extension of the half-gate technique. The communication cost of Ben-Efraim's [6] selector gate is the same as in the straightforward construction, while that work manages to reduce the computation cost by approximately 33% (i.e., 2 decryptions instead of 3 at evaluation time). We propose an *alternative designated selector gate*, which while it requires again 3 decryptions at evaluation time, it *reduces the communication cost to almost half* of that of Ben-Efraim's solution. We achieve this by making use of preprocessed data called daBits, proposed by Rotaru and Wood [24] and improved on in [1].

The other contribution of this work is to show how to perform multiparty garbling of both arithmetic circuits with *active security in the full-threshold multiparty setting*. We achieve this by using an authentication subprotocol akin to those in MASCOT [16] and in SPD$\mathbb{Z}_{2^k}$ [11] to apply the Boolean circuit garbling approach by Hazay et al. [14] to arithmetic garbling. One can view our

contribution as extending the work of Hazay et al. [14] to the arithmetic case and combining it with recent arithmetic garbling techniques.

2 Preliminaries

Security Model. The protocols in this work are proved secure in the universal composability (UC) framework of Canetti [10]. We consider an active, static adversary that can corrupt up to $n-1$ out of the n total parties. An active adversary may deviate arbitrarily from the protocol description, and a static adversary can choose which parties it will corrupt at the beginning of the protocol execution but not thereafter. Consequently, the functionalities are assumed to know at the beginning of their execution the set of corrupt parties: in the more general setting, the ideal-world adversary sends special "corruption" messages so that the functionality knows how to interact with different parties. Security is parameterised by the statistical security parameter, σ, and the computational security parameter, κ. We do not provide an implementation but typically one sets $\kappa \in \{64, 96, 128\}$ and $\sigma \in \{40, 80\}$ with $\sigma < \kappa$. We will make use of the standard functionalities $\mathcal{F}_{\mathsf{Rand}}$ given in Figure 2 and $\mathcal{F}_{\mathsf{Commit}}$ given in Figure 3.

Functionality $\mathcal{F}_{\mathsf{Rand}}$

On input (Rand, X) from all parties, sample $x \xleftarrow{\$} X$ uniformly and send x to all honest parties and $\mathcal{S}$.

Fig. 2: Functionality $\mathcal{F}_{\mathsf{Rand}}$ for agreeing on random strings sampled uniformly from a specified domain.

Functionality $\mathcal{F}_{\mathsf{Commit}}$

The ideal-world adversary is denoted by $\mathcal{S}$ and the indexing set of corrupt parties by A.

Initialise On input $(\mathsf{Initialise}, \mathsf{sid})$ from all honest parties, initialise a database DB.

Commit On input $(\mathsf{Commit}, x, i, \mathsf{sid})$ from party P_i, sample $\mathsf{id}_{x,i}$, store $(\mathsf{id}_{x,i}, x)$ in DB, and send $\mathsf{id}_{x,i}$ to all honest parties and $\mathcal{S}$.

Open On input $(\mathsf{Open}, \mathsf{id}_{x,i}, \mathsf{sid})$ from all parties and $\mathcal{S}$, retrieve $(\mathsf{id}_{x,i}, x)$ in DB;

- If $i \in A$ then await a message OK or Abort from $\mathcal{S}$. If the message is OK then send x to all honest parties; otherwise, send the message Abort to all honest parties and halt.
- If $i \in [n] \setminus A$, then send x to all honest parties and $\mathcal{S}$ and continue.

Fig. 3: Standard commitment functionality.

Secret-Sharing. We use the notation $\langle x \rangle$ to denote that the secret x is additively shared amongst the n parties: that is, the dealer samples $\{x^i\}_{i=1}^{n-1}$ uniformly at random from $\mathbb{F}$, sets $x_n := x - \sum_{i=1}^{n-1} x^i$, and for each $i \in [n]$ sends x^i (i.e., the i^{th} additive share of x) to party P_i.

We denote an *authenticated* shared value x by $[\![x]\!]$, which means that x is shared as above, and additionally there is some procedure for verifying that the sharing of x is not modified by the adversary. In the full-threshold setting, this is typically achieved by secret-sharing an information-theoretic Message Authentication Code (MAC) on every secret, as is done in BDOZ [8], TinyOT [22] and SPDZ [12]. The details of how secrets are authenticated in $\mathbb{F}_p$ and verified for correctness are not important for this work. If an error is introduced on any variable written as $[\![x]\!]$, this will be detected by the honest parties.

Garbling. We assume the reader is familiar with circuit garbling, but provide an overview here. A garbled circuit is a randomised version of a circuit that allows multiple parties to evaluate a function on the union of their private inputs without revealing anything more about their private inputs than what can be inferred from their own inputs and the output alone. In the two-party setting, this procedure is asymmetric; the high-level idea is as follows: one party, called the garbler, generates a "garbled" version of a circuit, hardwiring its own inputs in the circuit; then the other party, called the evaluator, evaluates the garbled circuit on its inputs (given some encoding information by the garbler that is provided in such a way that the garbler does not learn the evaluator's inputs) to obtain a "garbled" encoding of the output. At the end, the two parties communicate to reveal the final output to both.

Now we make things more concrete. Each fan-in-2 gate $g : \mathbb{F}^2 \to \mathbb{F}$ in the circuit with input wires u and v and output wire w is expressed as a table with one row for each $(\alpha, \beta) \in \mathbb{F}^2$ so that a row in the table has the form $(\alpha, \beta, g(\alpha, \beta))$. The garbler then samples a key for each possible value of α, β and $\gamma := g(\alpha, \beta)$. These keys typically live in some finite extension of the base field $\mathbb{F}^\ell$ where ℓ is $O(\kappa)$ so that the keys live in $O(2^\kappa)$, but general garbling does not prescribe how these keys should look except that certain garbling optimisations constrain the encryption scheme to have certain properties. The values in the input/output table are replaced with their corresponding encryption keys. Finally, the keys corresponding to the output wire w of the table are encrypted first under the key corresponding to the input on wire u input, and then under the key corresponding to the input on wire v input. In practice, the encryption function is a pseudorandom one-time-pad using a pseudorandom function (PRF) taking two keys, and using the gate index as a nonce so that the entry for input (α, β) in

the table representation of gate g is converted to a ciphertext:

$$\widetilde{g}_{\alpha,\beta} := F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + \mathsf{k}_{w,g(\alpha,\beta)},$$

where g is a gate index and acts as a nonce for the encryption, and $\mathsf{k}_{w,g(\alpha,\beta)}$ is the key. All of these $|\mathbb{F}|^2$ ciphertexts (i.e., the final column of the table) are handed to the evaluator. To begin evaluating, the evaluator is handed keys corresponding to its inputs and decrypts gates by computing $\widetilde{g}_{\alpha,\beta} - F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g)$. This results in a key that can be used to decrypt the next gate in the circuit (after the evaluator has also obtained the output key of another gate from elsewhere in the circuit). The evaluation involves proceeding iteratively through the circuit in this way, decrypting using pairs of keys, until a final output key is obtained.

To hide the inputs of the evaluator from the garbler when obtaining the initial gate input keys, the keys are sent using oblivious transfer (OT). Oblivious transfer is a channel in which a sender sends many messages, and the receiver selects one, with the guarantees that the sender cannot know which option the receiver selected and the receiver learns nothing about the messages it did not pick. In circuit garbling, for each wire on which the evaluator has input, the garbler sends the $|\mathbb{F}|$ different possible keys and the evaluator chooses the one corresponding to its input.

The circuit has the values of the garbler hardwired in. This is achieved, for example, by only encrypting under the "v" keys if the garbler provides the input on wire u for a given gate. However, this way the *order* of the ciphertexts may reveal to the evaluator the input of the garbler. To hide the garbler's input from the evaluator, the ciphertexts are randomly permuted using so-called *permutation* or *masking* values chosen by the garbler. In the arithmetic case, this is a rotation of the table rows. In order to evaluate the gates correctly, when evaluating a gate, in addition to learning the output key, the evaluator must learn a so-called *external* or *signal* value, which is the real value v masked with the masking value λ, that is, $e := v + \lambda$, so that it knows which ciphertexts to decrypt for each gate despite the rows being permuted. The ciphertexts are then

$$\widetilde{g}_{\alpha,\beta} := F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + \left(\mathsf{k}_{w,g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w} \middle\| (g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w\right),$$

where $g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w$ is the masked output wire (i.e., external) value. (The reader should think of the key as being in $\mathbb{F}^\ell$ for some ℓ of size $O(\kappa)$, and the external value as being in $\mathbb{F}$, and $F : \mathbb{F}^\ell \times \mathbb{F}^\ell \times \{0,1\}^{\log_2(|g|)} \to \mathbb{F}^{\ell+1}$.) The reader is referred to the original work of Beaver et al. [5] for a complete discussion of the permutation method (known as *point-and-permute*).

A technique known as FreeXOR, generalised for arithmetic circuits by Ben-Efraim et al. [7], can be employed to allow linear gates to be evaluated for free:

the garbler chooses a global difference R and then for every non-linear gate, the wire key for the value 0 is a random element $\mathsf{k}_{w,0}$ of $\mathbb{F}$ and the wire key for each value $\gamma \in \mathbb{F}_p \setminus \{0\}$ is set to $\mathsf{k}_{w,\gamma} := \mathsf{k}_{w,0} + \gamma \cdot R$. Then for linear (i.e., addition) gates, the output 0 wire key is defined as $\mathsf{k}_{w,0} := \mathsf{k}_{u,0} + \mathsf{k}_{v,0}$ and the corresponding mask as $\lambda_w := \lambda_u + \lambda_v$. Other gates are computed as:

$$\begin{aligned}\widetilde{g}_{\alpha,\beta} :=& F_{\mathsf{k}_{u,\alpha},\mathsf{k}_{v,\beta}}(g) + \\ &+ \Big(\mathsf{k}_{w,0} + (g(\alpha - \lambda_u, \beta - \lambda_v) + \lambda_w R) \Big\| (g(\alpha - \lambda_u, \beta - \lambda_v) + \lambda_w)\Big)\,.\end{aligned}$$

Note that instead of encrypting a concatenation of the masking bit with the key, the garbler can use a form of authenticated encryption, and then the evaluator decrypts ciphertexts until it finds a valid decrypted message and considers this the output key. This technique will be used in the garbling described later.

Half Gates. During the evaluation of the circuit, the signal values learnt by the evaluator "contain" the real values (in the sense that they are linearly dependent on them); likewise, the keys contain information regarding the real values. The idea behind half-gates is to exploit this information to reduce the amount of garbling required: during evaluation, the evaluator can compute the product of a signal value e_u with a key k_{v,e_v} to obtain "almost" a key for the product $v_u \cdot v_v$, and then can correct the errors that arise from the masking values using garbled gates (i.e., ciphertexts) in the more usual way[4]. In a sense, the difficult part of the multiplication gate, namely the cross-term $v_u \cdot v_v$ in the output key $\mathsf{k}_{w,e_w} = \mathsf{k}_{w,0} + (\lambda_w + v_u v_v)R$, is computed by computing $e_u \cdot \mathsf{k}_{v,e_v}$. The reason this is useful is that the errors that must be corrected in the product are each functions in the value of only *one* of the two real wire values v_u or v_v (and a combination of the (fixed) masking values). This means that the ciphertexts containing the corrections can be generated independently for each pair of inputs in $\mathbb{F}_p^2$ into the gate, which means only $p + p$ ciphertexts are needed, rather than $p \cdot p$ as required by garbling in the conventional manner.

To design a half gate, one observes what can be obtained from products of signal value with keys of input wires, namely from $e_u \cdot \mathsf{k}_{v,e_v}$, or from $e_v \cdot \mathsf{k}_{u,e_u}$. For

[4] This is analogous to the key-switching operation required for relinearisation of ciphertexts in somewhat-homomorphic encryption (SHE) schemes, where one first does a "naïve" multiplication, and then corrects the errors.

example,

$$\begin{aligned} e_u \mathsf{k}_{v,e_v} &= (v_u + \lambda_u)(\mathsf{k}_{v,0} + (v_v + \lambda_v)R) \\ &= v_u \mathsf{k}_{v,0} + \lambda_u \mathsf{k}_{v,0} + v_u v_v R + \lambda_u v_v R + v_u \lambda_v R + \lambda_u \lambda_v R \\ &= v_u v_v R + \underbrace{v_u \mathsf{k}_{v,0} + v_u \lambda_v R}_{\text{Dependent on } v_u} + \underbrace{\lambda_u v_v R}_{\text{Dependent on } v_v} + \underbrace{\lambda_u \mathsf{k}_{v,0} + \lambda_u \lambda_v R}_{\text{Dependent on neither}} \end{aligned}$$

Now since the goal is to obtain $\mathsf{k}_{w,e_w} = \mathsf{k}_{w,0} + (\lambda_w + v_u v_v)R$, for every $\gamma \in \mathbb{F}_p$ the garbler generates two ciphertexts: one encrypting

$$\mathsf{k}_{w,\mathsf{g},0} + \lambda_w R - ((\gamma - \lambda_u)(\mathsf{k}_{v,0} + \lambda_v R) + (\lambda_u \mathsf{k}_{v,0} + \lambda_u \lambda_v R)) \,,$$

and the other encrypting

$$\mathsf{k}_{w,\mathsf{e},0} - (\gamma - \lambda_v)(\lambda_u R) \,.$$

The output wire key is set to $\mathsf{k}_{w,0} := \mathsf{k}_{w,\mathsf{g},0} + \mathsf{k}_{w,\mathsf{e},0}$. The evaluator will decrypt the ciphertexts corresponding to $\gamma = e_u$ for the first half gate and $\gamma = e_v$ for the second; since $e_u - \lambda_u = v_u$ and $e_v - \lambda_v = v_v$, they will obtain the correct key by summing the two resulting plaintexts and the value $e_u \mathsf{k}_{v,e_v}$. Note that in the original two-party protocols, one gate input was assumed to come from the garbler and the other from the evaluator, so the evaluator would also be involved in the garbling of the half gates. This results in reduced communication since each party knows one of the wire masks. In the multiparty setting described later, no party knows the wire masks, so the main saving comes from reducing the quadratic cost p^2 to the linear cost $2 \cdot p$.

Some recent papers evaluate over a ring of primorial modulus rather than over a prime field in order to reduce the size of multiplication gates from $(\sum_{i=1}^{t} p_i)^2$ to $\sum_{i=1}^{t} p_i^2$ total ciphertexts. However, using the half-gate technique, the cost is the same regardless of the modulus, at $2 \cdot \sum_{i=1}^{t} p_i$ ciphertexts. The CRT approach is also useful for performing non-linear operations such as computing powers. These operations are quite expensive even in the passive security setting. While it may be useful to have an actively-secure protocol for arithmetic circuits over a composite modulus ring, there are difficult challenges to overcome arising from the presence of zero divisors; thus we leave this to future work.

We evaluate the garbled circuits in $\mathbb{F}_p$, for which the straightforward garbling approach requires that p be small enough to allow parties to send $O(p)$ ciphertexts per multiplication gate, but large enough so that the PRF keys used for encryption are computationally secure. To do this, we evaluate circuits in $\mathbb{F}_p$, but take keys in an extension field, specifically $\mathbb{F}_{p^{\ell_\kappa}}$, where $\ell_\kappa := 1 + \lceil \kappa / \log p \rceil$.

Multiparty Garbling. In multiparty garbling, originally developed by Beaver et al. [5], all parties act as garbler and evaluator. Lindell et al. [19] showed how to use actively-secure secret-sharing-based MPC to compute a multiparty garbled circuit with active security. Using MPC, each party generates keys for a circuit, and the masking values are chosen randomly and are unknown to the parties. This way for each gate, each party holds n ciphertexts, indexed by j:

$$\widetilde{g}^j_{\alpha,\beta} := \sum_{i=1}^{n} F_{\mathsf{k}^i_{u,\alpha},\mathsf{k}^i_{v,\beta}}(g,j) + \left(\mathsf{k}^j_{w,0} + (g(\alpha-\lambda_u,\beta-\lambda_v)+\lambda_w)R^j\right).$$

Since each party P_i generates one set of keys (those indexed by i), the external values on the wires can be learnt by each party examining the output plaintext m^i from its own circuit and setting $e_w := (m^i - \mathsf{k}^i_{w,0}) \cdot {R^i}^{-1}$ and $\mathsf{k}^i_{w,e_w} := m^i$.

In many ways, the protocol we present in this work is a straightforward generalisation of garbling protocols over $\mathbb{F}_2$. Notice that for a Boolean circuit, the half-gate approach is no more efficient than the naïve approach, unless we are in the two-party setting in which one party is the garbler and one the evaluator, rather than all being both as in the multiparty setting.

PRF Assumption. To encrypt a gate, a single-keyed PRF is evaluated on a nonce and used to one-time-pad encrypt a key. To make use of the (generalised) FreeXOR technique, the following assumption is required.

Let $F : \mathbb{F}_{p^{\ell_\kappa}} \times \mathbb{N} \to \mathbb{F}_{p^{\ell_\kappa}}$ be a keyed pseudorandom function (PRF). Define the oracle $\mathcal{O}_{F,R}$ in the following way:

$$\begin{aligned}\mathcal{O}_{F,R} : \mathbb{F}_{p^{\ell_\kappa}} \times \mathbb{F}_p \times \mathbb{N} \times \mathbb{F}_p &\to \mathbb{F}_{p^{\ell_\kappa}} \\ \mathcal{O}_{F,R}(\mathsf{k},\gamma,x,\delta) &\mapsto F_{\mathsf{k}+\gamma\cdot R}(x) + \delta \cdot R\end{aligned}$$

Now define $\mathcal{F}_{\mathsf{RO}}$ to be an oracle that, on input a query $m = (\mathsf{k},\gamma,x,\delta) \in \mathbb{F}_{p^{\ell_\kappa}} \times \mathbb{F}_p \times \mathbb{N} \times \mathbb{F}_p$, if m has not been queried before, samples $r \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and outputs r, and otherwise outputs whatever was sampled previously.

The following definition was given by Hazay et al. [14] for Boolean functions, and a similar definition for arithmetic circuits was given by Ball et al. [4].

Definition 1 (Circular Correlation Robustness). *For the oracles above, define* legal queries *as those with inputs in the correct domain, and additionally:*

1. *The oracle may not be queried when $\gamma = 0$.*
2. *The oracle may not be queried twice for the same δ unless at least one other variable changes.*

Then we say that F is circular correlation robust *if for all probabilistic polynomial-time distinguishers $\mathcal{D}$, it holds that*

$$\left| \Pr_{R \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}} [\mathcal{D}^{\mathcal{O}_{F,R}}(1^\kappa)] - \Pr[\mathcal{D}^{\mathcal{F}_{\mathsf{RO}}}(1^\kappa)] \right| = O(2^{-\kappa})$$

In the garbling protocols, the PRF is queried on values (g, j), where $g \in \mathbb{N}$ is the gate index and $j \in [n]$ is the party index, parsed as a natural number $\lceil \log n / \log 10 \rceil \cdot g + j$.

The choice for this definition comes from the fact that parties should not be able to distinguish between keys generated using global differences and uniform keys in the field. Note that while the keys generated for each wire are only in some coset $\mathsf{k}_{w,0} + \{\gamma R : \gamma \in \mathbb{F}_p\}$ of $\mathbb{F}_{p^{\ell_\kappa}}$, the distinguisher is only allowed to query once per key per nonce for a fixed δ. This corresponds to the fact that in the garbling, the evaluator(s) can only decrypt a single ciphertext.

3 Full-Threshold Active Security

We define an n-party arithmetic garbling protocol by extending the state-of-the-art techniques used by Hazay et al. [14] for Boolean garbling to arithmetic garbling, using actively-secure MPC over $\mathbb{F}_p$ as a black box, and using the half-gate techniques described for arithmetic circuits by Ben-Efraim [6]. In this section we describe the actively-secure garbling of the "standard" multiplication gate, since using the classical garbling techniques one can replace the multiplication function with any gate $g : \mathbb{F}_p^2 \to \mathbb{F}_p$; our techniques for active security also apply to other gates, and indeed in the protocol later we garble multiplication half gates. Many of the techniques due to Hazay et al. [14] apply almost immediately to the arithmetic case and so the exposition here closely follows theirs. We will first explain the components of the garbling protocol at a high level, then discuss how to realise these different parts, and finally we will give the complete protocol.

3.1 Overview

In the arithmetic analogue of the multiparty garbling protocol of Beaver et al. [5], with the optimisations of Section 2, we aim to produce a set of $p^2 \cdot n$ ciphertexts, indexed by $j \in [n]$ and $(\alpha, \beta) \in \mathbb{F}_p^2$, for each multiplication gate, of the form:

$$\widetilde{g}^j_{\alpha,\beta} := \left(\sum_{i=1}^{n} F_{\mathsf{k}^i_{u,\alpha}, \mathsf{k}^i_{v,\beta}}(g, j) \right) + \mathsf{k}^j_{w,0} + R^j \cdot ((\alpha - \lambda_u) \cdot (\beta - \lambda_v) + \lambda_w),$$

where the wire masks λ_u, λ_v and λ_w are not known to any party and the keys indexed by i are generated by P_i. For now, the reader can think of $\mathsf{k}_{u,\alpha}$, $\mathsf{k}_{v,\beta}$, $\mathsf{k}_{w,0}$ and R^j as lying in a finite extension of $\mathbb{F}_p$ – the same space as the codomain of the PRF. The approach of Hazay et al. for Boolean circuits to produce these ciphertexts with active security is to generate a secret-shared version of $\widetilde{g}^j_{\alpha,\beta}$ for every $j \in [n]$ and open them, in the following way:

1. Use a generic "Bit-MPC" functionality, $\mathcal{F}_{\mathsf{BitMPC}}$, for parties to obtain authenticated secret-shared random bits $[\![\lambda_u]\!]$, $[\![\lambda_v]\!]$ and $[\![\lambda_w]\!]$ and to compute $[\![\lambda_u \cdot \lambda_v]\!]$.
2. Use correlated oblivious transfer (COT) to compute the products by the global differences: for each $j \in [n]$ to compute secret-shared versions of:

$$R^j \cdot \lambda_u, \quad R^j \cdot \lambda_v, \quad R^j \cdot (\lambda_w + \lambda_u \cdot \lambda_v).$$

3. *Locally* combine the secret-shared values with local PRF evaluations to obtain a sharing of each gate $\widetilde{g}^j_{\alpha,\beta}$.
4. Open all the sharings.

A key observation, first made by Lindell et al. [19], is that the sharings need not be authenticated, as the parties will abort during circuit evaluation with overwhelming probability if the adversary introduces errors. This means that the PRF evaluations need neither be authenticated, nor proved correct using a zero-knowledge proof. Authentication is required on the wire masks to ensure the multiplication is performed correctly. Thus, only *one* secure Bit-MPC multiplication is required per AND gate, along with an amortised COT operation.

Our approach here is to give the simple generalisation for the field $\mathbb{F}_p$, noting that the keys must live in the space $\mathbb{F}_{p^{\ell_\kappa}}$, where $\ell_\kappa := 1 + \lceil \kappa / \log p \rceil$. We first describe the replacement of $\mathcal{F}_{\mathsf{BitMPC}}$ with MPC over a field, denoted by $\mathcal{F}_{\mathsf{MPC}}$, and second show how to replace COT with correlated oblivious product evaluation (COPE) (also known as vector oblivious linear function evaluation (vOLE)).

3.2 Secret-Sharing-Based Wire Mask Arithmetic

For arithmetic circuits, the bit masks are replaced with masks in $\mathbb{F}_p$ and the functionality $\mathcal{F}_{\mathsf{BitMPC}}$ is replaced with $\mathcal{F}_{\mathsf{MPC}}$, shown in Figure 4. Instead of *any* generic $\mathcal{F}_{\mathsf{MPC}}$ functionality, we model here a secret-sharing-based functionality, which can be instantiated with any actively secure protocol for secret-sharing-based arithmetic MPC. We denote by $[\![x]\!]$ an authenticated secret shared value x that is stored internally by $\mathcal{F}_{\mathsf{MPC}}$. Then, x^i denotes party P_i's additive share of x. In the garbling protocol, just as in the work of Hazay et al. [14], to obtain a wire mask λ_u, each party samples $\lambda_u^i \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ to create an authenticated sharing of this value; then they call **Add** to obtain $[\![\lambda_u]\!] = \sum_{i=1}^n [\![\lambda_u^i]\!]$. They do

similarly for λ_v and λ_w so that the parties obtain $[\![\lambda_u]\!]$, $[\![\lambda_v]\!]$ and $[\![\lambda_w]\!]$, and then call **Multiply** to multiply $[\![\lambda_u]\!]$ and $[\![\lambda_v]\!]$, and obtain $[\![\lambda_{uv}]\!] = [\![\lambda_u \cdot \lambda_v]\!]$.

Functionality $\mathcal{F}_{\mathsf{MPC}}$

The functionality assumes n parties $P_1, \ldots, P_n$, and the ideal-world adversary $\mathcal{S}$, who controls a subset of parties $I \subset [n]$. If a command is received where sid differs from what was sent during **Initialise**, ignore the command and await the next. At any point, the ideal-world adversary $\mathcal{S}$ can send the message Abort and the functionality sends the message Abort to all honest parties and halts.

Initialise On input $(\mathsf{Initialise}, \mathbb{F}, \mathsf{sid})$, store the field $\mathbb{F}$ and initialise a new database, $\mathsf{DB} := \varnothing$.

Input On input $(\mathsf{Input}, i, \mathsf{id}_1, \ldots, \mathsf{id}_\ell, x_1, \ldots x_\ell, \mathsf{sid})$ from P_i and $(\mathsf{Input}, i, \mathsf{id}_1, \ldots \mathsf{id}_\ell, \bot, \mathsf{sid})$ from all other parties, where $x_i \in \mathbb{F}$ and id_i are distinct new identifiers, append the ℓ entries (id_i, x_i) to DB.

Add On input $(\mathsf{Add}, \mathsf{id}_x, \mathsf{id}_y, \mathsf{id}_z, \mathsf{sid})$ from all parties, where id_x and id_y are identifiers in the database and id_z is a new identifier, retrieve (id_x, x) and (id_y, y) from memory and append the entry $(\mathsf{id}_z, x + y)$ to DB.

Multiply On input $(\mathsf{Multiply}, \mathsf{id}_x, \mathsf{id}_y, \mathsf{id}_z, \mathsf{sid})$ from all parties, where id_x and id_y are identifiers in the database and id_z is a new identifier, retrieve (id_x, x) and (id_y, y) from memory and compute $z := x \cdot y$. Receive shares $z^i \in \mathbb{F}$ from $\mathcal{S}$, for $i \in I$, randomly sample honest parties' shares $z^j \in \mathbb{F}$ for $j \notin I$ s.t. $\sum_{i=1}^{n} z^i = z$, send z^i to $P_i, i \in [n]$, and append (id_z, z) to DB.

Output On input $(\mathsf{Output}, \mathsf{id}_x, \mathsf{sid})$ from all parties, retrieve the entry (id_x, x) from DB, send x to $\mathcal{S}$, and await a message OK or Abort; if the message is OK then send x to all honest parties, and otherwise send the message Abort to all honest parties and halt.

Fig. 4: Functionality $\mathcal{F}_{\mathsf{MPC}}$ for performing general MPC, secure *with abort*.

3.3 Wire Mask/Global Difference Products

In the garbling protocol, for every wire w the parties require (unauthenticated) sharings of $R^j \cdot \lambda_w$ for every $j \in [n]$. Since λ_w is additively shared, the parties actually compute sharings of $R^j \cdot \lambda_w^i$ for every $j \in [n]$ and $i \neq j$. Since the global difference is fixed for all gates in the circuit, in the Boolean case such sharings can be generated using COT, in which a sender chooses a fixed correlation, namely R^j, and the receiver inputs their sharing of the mask λ_w^i; then the sender obtains some $q^{j,i}$ and the receiver some $t^{i,j}$ such that $q^{j,i} + t^{i,j} = \lambda_w^i \cdot R^j$. Hazay et al.'s [14] protocol for computing the wire mask/global difference products is called $\Pi_{\mathsf{Bit} \times \mathsf{String}}$, since $R^j \in \mathbb{F}_{2^k}$ and the masks are bits.

We can apply essentially the same techniques here, and correctness of the protocol follows in exactly the same way. The difference is that we are now

interested in masks in $\mathbb{F}_p$ and global differences in $\mathbb{F}_{p^{\ell_\kappa}}$. Thus, we must use the correlated oblivious product evaluation (COPE) presented in Figure 5, which is an extension of the protocol $\Pi_{\mathsf{Bit}\times\mathsf{String}}$ [14], operating in any finite field, instead of only in $\mathbb{F}_2$. Note that $\mathcal{F}_{\mathsf{COPE}}$ accepts inputs from the sender in $\mathbb{F}_{p^{\ell_\kappa}}$, but in our protocol the inputs are assumed to be in $\mathbb{F}_p$, as they are circuit wire masks. Thus a corrupt sender could send an element of $\mathbb{F}_{p^{\ell_\kappa}} \setminus \mathbb{F}_p$ in the instance of $\mathcal{F}_{\mathsf{COPE}}$. However, the follow-up checks that take place during the execution of the subprotocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ (Fig. 6) ensure that secrets lie in $\mathbb{F}_p$. A functionality such as $\mathcal{F}_{\mathsf{OLE}}^{t,1}$ by Ghosh et al. [13] that accepts input from the sender in a small field, from the receiver in an extension field and outputs a sharing in the larger field could be used, but for a technical reason it is not amenable to OT extension [15] as is $\mathcal{F}_{\mathsf{COPE}}$ and is therefore less efficient when performing a large number of multiplications. Realising a product functionality more efficiently would improve the overall efficiency of the garbling protocol and we leave this for future work.

Functionality $\mathcal{F}_{\mathsf{COPE}}$ (from [16])

Let $\mathbf{g} : \mathbb{F}^{\lceil \log |\mathbb{F}| \rceil} \to \mathbb{F}$ be any map such that for every $x \in \mathbb{F}$, if $\mathbf{x} \in \{0,1\}^{\lceil \log |\mathbb{F}| \rceil}$ represents its bit-decomposition, then $\mathbf{g}(\mathbf{x}) = x$. Let $\mathbf{g}^{-1}(x)$ denote the bit-decomposition of x, which is well-defined by uniqueness of decomposition.

Initialise On receiving the message $(\mathsf{Initialise}, \mathbb{F}, P_j, P_i, \mathsf{sid}_{j,i})$ from parties P_i and P_j, await $\Delta \in \mathbb{F}$ from P_j, store Δ, and set $\boldsymbol{\Delta} := \mathbf{g}^{-1}(\Delta)$.

Extend On receiving the message $(\mathsf{Extend}, \mathsf{sid}_{j,i})$ from both parties,

1. – If P_i and P_j are honest then await $x \in \mathbb{F}$ from P_i, sample $q \xleftarrow{\$} \mathbb{F}$ and set
$$t = x \cdot \Delta - q$$
– If P_i is corrupt and P_j is honest then await $t \in \mathbb{F}$ and $\mathbf{x} \in \mathbb{F}^{\lceil \log |\mathbb{F}| \rceil}$ from $\mathcal{S}$ and set
$$q = \mathbf{g}(\mathbf{x} * \boldsymbol{\Delta}) - t$$
where $*$ denotes the coordinatewise product.
– If P_i is honest and P_j is corrupt then await $x \in \mathbb{F}$ from P_i and $q \in \mathbb{F}$ from $\mathcal{S}$ and compute
$$t := x \cdot \Delta - q.$$
2. Send t to P_i and q to P_j.

Fig. 5: Functionality for Correlated Oblivious Product Evaluation.

The subprotocol for mutliplying global differences with wire masks is given in Figure 6.

Subprotocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$

Initialise For every ordered pair of parties (P_j, P_i), call an instance of $\mathcal{F}_{\mathsf{COPE}}$, denoted by $\mathcal{F}_{\mathsf{COPE}}^{(j,i)}$ with P_i as the sender and P_j as the receiver, with input $(\mathsf{Initialise}, \mathbb{F}_{p^{\ell_\kappa}}, P_j, P_i, \mathsf{sid}_{j,i})$, and input R^j from P_j.

Multiply To compute unauthenticated sharings $(\langle x_k \cdot R^i \rangle)_{k=1}^m$ from authenticated sharings $([\![x_k]\!])_{k=1}^m$ for which the parties additionally hold $(\langle x_k \rangle)_{k=1}^m$, the parties do the following:

1. **Mask** The parties generate $\ell_\sigma := \lceil \sigma / \log p \rceil$ masks: for each $l \in [\ell_\sigma]$,
   (a) For each $i \in [n]$, party P_i samples $x_{m+l}^i \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ with input $(\mathsf{Input}, i, x_{m+l}^i, \mathsf{id}_{x_{m+l}^i})$ while each party P_j, $j \neq i$, provides corresponding input $(\mathsf{Input}, i, \bot, \mathsf{id}_{x_{m+l}^i})$.
   (b) The parties obtain $[\![x_{m+l}]\!] = \sum_{i=1}^n [\![x_{m+l}^i]\!]$ by creating a new identifier $\mathsf{id}_{x_{m+l}}$ and calling the **Add** procedure of $\mathcal{F}_{\mathsf{MPC}}$ multiple times.
2. **Generate** For each $j \in [n]$,
   (a) For every $i \neq j$,
      i. P_i and P_j call $\mathcal{F}_{\mathsf{COPE}}^{(i,j)}$ with input $(\mathsf{Extend}, \mathsf{sid}_{i,j})$:
         A. P_i provides $x_1^i, \ldots, x_{m+l}^i$ as input.
         B. P_i receives $(t_k^{i,j})_{k=1}^{m+l}$ and P_j receives $(q_k^{j,i})_{k=1}^{m+l}$.
      ii. It holds that $q_k^{j,i} + t_k^{i,j} = x_k^i R^j$. Party P_i sets $z_k^{i,j} := t_k^{i,j}$.
   (b) Party P_j sets $z_k^{j,j} := x_k^j R^j + \sum_{i \neq j} q_k^{j,i}$.
3. **Check**
   (a) Call $\mathcal{F}_{\mathsf{Rand}}$ with input $(\mathsf{Rand}, \mathbb{F}_p^{\ell_\sigma \times m})$ to obtain a matrix $H = (\chi_{l,k})_{l \in [\ell_\sigma], k \in [m]}$.
   (b) Let $\mathbf{x} := (x_k)_{k=1}^m$ and $\hat{\mathbf{x}} := (x_{m+l})_{l=1}^{\ell_\sigma}$. The parties compute $[\![\mathbf{c}]\!] := H \cdot [\![\mathbf{x}]\!] + [\![\hat{\mathbf{x}}]\!]$ and call $\mathcal{F}_{\mathsf{MPC}}$ with input $(\mathsf{Output}, \mathsf{id}_{\mathbf{c}}, \mathsf{sid})$ to obtain $\mathbf{c}$. If it aborts, then the parties abort.
   (c) Each party P_i computes $\mathbf{c}^{i,j} := H \cdot (z_k^{i,j})_{k=1}^m + (z_{m+l}^{i,j})_{l=1}^{\ell_\sigma}$ and $\mathbf{c}^{i,i} := -\mathbf{c} \cdot R^i + H \cdot (z_k^{i,i})_{k=1}^m + (z_{m+l}^{i,i})_{l=1}^{\ell_\sigma}$.
   (d) Each party P_i calls $\mathcal{F}_{\mathsf{Commit}}$ with input $(\mathsf{Commit}, \mathbf{c}^{i,j}, i, \mathsf{sid})$ for all $j \in [n]$.
   (e) When $\mathsf{id}_{\mathbf{c}^{i,j}}$ has been received from $\mathcal{F}_{\mathsf{Commit}}$ for all $i, j \in [n]^2$, call $\mathcal{F}_{\mathsf{Commit}}$ with input $(\mathsf{Open}, \mathsf{id}_{\mathbf{c}^{i,j}}, \mathsf{sid})$.
   (f) Check that $\sum_{i=1}^n \mathbf{c}^{i,j} = \mathbf{0}$ for all $j \in [n]$. If so, then each party P_i (locally) outputs $(z_k^{i,j})_{k \in [m], j \in [n]}$; otherwise, they abort.

Fig. 6: Subprotocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ for multiplying global differences with wire masks.

For active security, it is necessary to check that each P_j provides the same global difference R^j with every other P_i, and that every P_i provides the same sharing λ_w^i with every other P_j. Observe that

$$\left(x^j \cdot R^j + \sum_{i \neq j} q^{j,i}\right) + \left(\sum_{i \neq j} t^{i,j}\right) = R^j \cdot \left(x^j + \sum_{j \neq i} (q^{j,i} + t^{i,j})\right) = R^j \cdot \left(\sum_{i=1}^n x^i\right)$$

where the first summand is computed by party P_j and for each $i \neq j$, $t^{i,j}$ is held by P_i. The fact that this relationship must hold (by design) can be used to check correctness of a *batch* of secrets $\{[\![x_k]\!]\}_{k=1}^m$ as follows: parties can take an additional mask $[\![x_{m+1}]\!]$, reveal a random linear combination $c := x_{m+1} + \sum_{k=1}^m \chi_k x_k, \chi_k \in \mathbb{F}_p \forall k$, and check for all $j \in [n]$ that $\langle z^j \rangle$ defined by

$$z^{i,j} := t^{i,j}_{m+1} + \sum_{k=1}^{m} \chi_k \cdot t^{i,j}_k \qquad (i \neq j)$$

and

$$z^{j,j} := -c \cdot R^j + \left(x^j_{m+1} \cdot R^j + \sum_{i \neq j} q^{j,i}_{m+1} \right) + \sum_{k=1}^{m} \chi_k \cdot \left(x^j_k \cdot R^j + \sum_{i \neq j} q^{j,i}_k \right)$$

is an additive sharing of 0. It will be shown in the proof of Lemma 1 that the probability that parties are inconsistent but all of the n sharings $\{\langle z^i \rangle\}_{i=1}^n$ are zero is bounded above by p^{-1}; thus the check is performed independently $\ell_\sigma := \lceil \sigma / \log p \rceil$ times in parallel to ensure at least σ bits of statistical security.

Concrete instantiation. One of the reasons that the protocol of Hazay et al. [14] is so efficient is that the functionality $\mathcal{F}_{\mathsf{BitMPC}}$ can be realised using the n-party variant [9] of the TinyOT [22] protocol, in which bits are authenticated exactly via sharings of $b^i \cdot R^j$, where R^j is taken to be the secret key of P_j. Thus sharings of the wire mask/global difference products are immediately available to the parties by the correctness of the $\mathcal{F}_{\mathsf{BitMPC}}$ functionality, without the need for a separate $\Pi_{\mathsf{Bit}\times\mathsf{String}}$ protocol. However, currently the most efficient protocols in the setting of a large prime field use a different form of authentication and so this optimisation cannot be directly applied here. Instead, we can use, for example, the most recent version of the SPDZ protocol [12] known as Overdrive [17]. Note that in MASCOT [16], pairwise MACs are generated and then combined to create global MACs, so it may be that this approach, which then obviates the need to perform the protocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ separately, is better in practice.

Lemma 1, states that an adversary succeeds in cheating without detection in $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ with negligible probability in the statistical security parameter, σ.

Lemma 1. *For the outputs* $(z^{i,j}_k)_{i,j\in[n]}$ *of the subprotocol* $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ *it holds that* $\sum_{i=1}^n z^{i,j}_k = x^i_k \cdot R^j$ *for all* j *except with probability at most* $2^{-\sigma}$.

Notice that in order to establish a unique signal value after decrypting ciphertexts, it is necessary to multiply by R^{-1}, which means that R must be invertible. However, since R is sampled from a field, the random choice is invertible except if it is 0, which happens with probability $p^{-\ell_\kappa} < 2^{-\kappa} < 2^{-\sigma}$.

Proof. The proof follows the same line as that of Hazay et al. [14, Lem 3.1] and Cramer et al. [11, Thm 3]. There are two possible ways to cheat in the protocol: (a) one or more parties use different inputs to the initialisation of $\mathcal{F}_{\mathsf{COPE}}$ with different honest parties; (b) one or more parties use different shares x_k^i with different honest parties. We show that if these errors are non-zero then the protocol aborts with overwhelming probability, by showing that the probability that the checks pass but a non-zero error has been introduced is negligible.

Fix an honest party P_{i^*} arbitrarily[5] and for each $i \neq i^*$ let R^i be the global difference provided by P_i into the instance $\mathcal{F}_{\mathsf{COPE}}^{i^*,i}$ and let x_k^i be the input of P_i into the instance $\mathcal{F}_{\mathsf{COPE}}^{i,i^*}$. Then for each $i \neq i^*$, for each $j \neq i$ let $\widetilde{R}^{j,i}$ be the input of P_j into the instance $\mathcal{F}_{\mathsf{COPE}}^{i,j}$, or the local contribution to $z_k^{j,j}$ when $i = j$, and let $\varepsilon^{j,i} := \widetilde{R}^{j,i} - R^j$. Similarly, for each $j \neq i^*$, for each $i \neq j$ let $\widetilde{x}_k^{j,i}$ be the input of P_j into the instance $\mathcal{F}_{\mathsf{COPE}}^{j,i}$, or the local contribution to $z_k^{j,j}$ when $i = j$, and let $\delta^{j,i} := \widetilde{x}_k^{j,i} - x_k^j$.

Let ε^j be the error introduced when creating $c^{j,j}$ and ε_l be the sum of the errors introduced just before the l^{th} commitment. Now since the checks pass, for all $j \in [n]$ and all $l \in [\ell_\sigma]$ it holds that

$$
\begin{aligned}
0 &= \sum_{i=1}^{n} c_l^{i,j} = c_l^{j,j} + \sum_{i \neq j} c_l^{i,j} \\
&= \left(\varepsilon_l - c_l R^j + \left(z_{m+l}^{j,j} + \sum_{k=1}^{m} \chi_{l,k} \cdot z_k^{j,j} \right) \right) + \sum_{i \neq j} \left(z_{m+l}^{i,j} + \sum_{k=1}^{m} \chi_{l,k} \cdot z_k^{i,j} \right) \\
&= \varepsilon_l - c_l R^j + \sum_{i=1}^{n} \left(z_{m+l}^{i,j} + \sum_{k=1}^{m} \chi_{l,k} \cdot z_k^{i,j} \right) \\
&= \varepsilon_l - c_l R^j + \sum_{i=1}^{n} z_{m+l}^{i,j} + \sum_{i=1}^{n} \sum_{k=1}^{m} \chi_{l,k} \cdot z_k^{i,j} \\
&= \varepsilon_l - c_l R^j + \left(\left(\widetilde{x}_{m+l}^{j,j} \widetilde{R}^{j,j} + \sum_{i \neq j} q_{m+l}^{j,i} \right) + \sum_{i \neq j} t_{m+l}^{i,j} \right) + \\
&\qquad \sum_{k=1}^{m} \chi_{l,k} \cdot \left(\left(\widetilde{x}_k^{j,j} \widetilde{R}^{j,j} + \sum_{i \neq j} q_k^{j,i} \right) + \sum_{i \neq j} t_k^{i,j} \right)
\end{aligned}
$$

[5] It can be shown that if there are multiple honest parties then the sum of the errors subsequently defined with respect to P_{i^*} is well-defined (i.e., the sum is independent of the choice of "reference" honest party).

$$
\begin{aligned}
&= \varepsilon_l - c_l R^j + \left(\left(\sum_{i=1}^{n} \widetilde{x}^{i,j}_{m+l} \widetilde{R}^{j,i} \right) + \left(\sum_{k=1}^{m} \chi_{l,k} \cdot \sum_{i=1}^{n} \widetilde{x}^{i,j}_{k} \widetilde{R}^{j,i} \right) \right) \\
&= \varepsilon_l - c_l R^j + \sum_{i=1}^{n} \left(\widetilde{x}^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} \widetilde{x}^{i,j}_{k} \right) \cdot \widetilde{R}^{j,i} \\
&= \varepsilon_l - \cancel{c_l R^j} + \cancel{\sum_{i=1}^{n} \left(x^{i}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} x^{i}_{k} \right) R^j} + \sum_{i=1}^{n} \left(\delta^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} \delta^{i,j}_{k} \right) R^j \\
&\qquad + \sum_{i=1}^{n} \left(x^{i}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} x^{i}_{k} \right) \varepsilon^{j,i} + \sum_{i=1}^{n} \left(\delta^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} \delta^{i,j}_{k} \right) \varepsilon^{j,i} \\
&= \varepsilon_l + \underbrace{\sum_{i=1}^{n} \left(\delta^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} \delta^{i,j}_{k} \right)}_{=:\delta^{j}_{l}} R^j + \sum_{i=1}^{n} \left(x^{i}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} x^{i}_{k} \right) \varepsilon^{j,i} \\
&\qquad + \sum_{i=1}^{n} \left(\delta^{i,j}_{m+l} + \sum_{k=1}^{m} \chi_{l,k} \delta^{i,j}_{k} \right) \varepsilon^{j,i}
\end{aligned}
$$

Suppose P_j is an honest party; then we aim to show that $\delta^{i,j}_k = 0$ for all $k \in [m]$, $i \in [n]$ and that $\varepsilon^{j,i} = 0$ for all $i \in [n]$, except with negligible probability in σ.

If $\delta^j_l \neq 0$ then since R^j is generated by the honest party, the second summand is uniform in $\mathbb{F}_{p^{\ell_\kappa}}$ and unknown to the adversary, so the adversary can choose the other errors so that the equation above holds with probability at most $p^{-\ell_\kappa}$.

If $\delta^j_l = 0$ then $\delta^{i,j}_k = 0$ for all $k \in [m]$ except with probability p^{-1}, since the coefficients are unknown before the errors are introduced; thus the fourth summand is also 0. The greatest upper bound on the probability is only p^{-1} and not $p^{-\ell_\kappa}$, because although these errors are in $\mathbb{F}_{p^{\ell_\kappa}}$, the coefficients lie in $\mathbb{F}_p$. Since the matrix H is sampled after the errors $\varepsilon^{j,i}$ are introduced on the global differences – i.e., the values $\chi_{l,k}$ are unknown to the adversary when introducing these errors – if one or more $\varepsilon^{j,i}$ is non-zero then the third summand is uniformly random in $\mathbb{F}_p$, so the adversary must choose each ε_l to correct the error, which can be done for a given l with probability at most p^{-1}. Thus the probability that some $\delta^{i,j}_k$ or some $\varepsilon^{j,i}$ is not 0 but the equation above holds for all $l \in [\ell_\sigma]$ is at most $p^{-\ell_\sigma}$. Thus for every honest party $j \in [n] \setminus A$, all errors $\{\delta^{i,j}_k\}$ and $\{\varepsilon^{j,i}\}$ are 0 except with probability at most $\max\{p^{-\ell_\kappa}, p^{-\ell_\sigma}\} < 2^{-\sigma}$. ■

3.4 The Complete Garbling and Evaluation Protocols

Following the analysis of the necessary components for garbling we now present the complete garbling and evaluation protocols. The subprotocol for garbling is given in Figure 7, and the one for evaluation in Figure 8.

Theorem 1. *The execution of the subprotocol Π_{Garble} followed by the execution of the subprotocol Π_{Eval}, making use of the subprotocol $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$, UC-securely realises the functionality $\mathcal{F}_{\mathsf{AC}}$ in the presence of a static, active adversary that corrupts up to $n-1$ parties, in the $\mathcal{F}_{\mathsf{Commit}}$,$\mathcal{F}_{\mathsf{COPE}}$,$\mathcal{F}_{\mathsf{MPC}}$,$\mathcal{F}_{\mathsf{Rand}}$-hybrid model, assuming the PRF F satisfies correlation-robustness.*

We define a security game in which a successful adversary breaks the circular correlation robustness assumption of a PRF F in the following way:

Game

1. The challenger $\mathcal{C}$ samples a bit $b \xleftarrow{\$} \{0,1\}$; if $b = 0$ then it initialises the oracle $\mathcal{O} := \mathcal{F}_{\mathsf{RO}}$; if $b = 1$ then it samples $R \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and initialises the oracle $\mathcal{O} := \mathcal{O}_{F,R}$.
2. The adversary $\mathcal{D}$ is provided with black-box access to $\mathcal{O}$.
3. After polynomially-many queries to $\mathcal{O}$, $\mathcal{D}$ outputs a bit b' to $\mathcal{C}$.
4. The adversary $\mathcal{D}$ wins the game if $b' = b$.

We are now ready to prove the theorem.

Proof. Note that, in contrast to the work of Hazay et al. [14], the output masks in our work are *not* revealed after garbling; instead, the parties remove the masks *after* evaluating the circuit. Opening the wire masks after the evaluation is a common approach, taken for example by Wang et al. in their recent multiparty Boolean garbling protocol [25] and leads to a more straightforward proof.

To prove that the protocol UC-securely realises the functionality $\mathcal{F}_{\mathsf{AC}}$ under the assumption that the PRF is correlation-robust, we will construct a simulator interacting with the real-world adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{\mathsf{AC}}$ such that if an environment can determine that $\mathcal{A}$ is interacting with $\mathcal{S}$ instead of real honest parties then it must have been able to break the assumption on the PRF. To do this, we will construct a situation in which distinguishing between worlds immediately leads to a way to break the PRF assumption. Consider the simulator $\mathcal{S}$ defined as follows:

1. Execute Π_{Garble} honestly, sampling keys, masks and global differences as honest parties would in the execution of $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ and honestly executing internal copies of the oracles $\mathcal{F}_{\mathsf{Commit}}$, $\mathcal{F}_{\mathsf{COPE}}$, $\mathcal{F}_{\mathsf{MPC}}$ and $\mathcal{F}_{\mathsf{Rand}}$. Recall that we have shown in the proof of Lemma 1 that $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ provides statistical security against an active adversary, except with probability at most $2^{-\sigma}$.

2. Sample inputs uniformly in $\mathbb{F}_p$ on behalf of emulated honest parties, compute signal values honestly, and send these signal values to $\mathcal{A}$; await the broadcasts from corrupt parties and extract the inputs using knowledge of the masks from the calls to $\mathcal{F}_{\mathsf{MPC}}$ in Step 1, and send the inputs to $\mathcal{F}_{\mathsf{AC}}$. Store the signal values for all of these input wires.
3. Await the final circuit output from $\mathcal{F}_{\mathsf{AC}}$.
4. Determine the "evaluation path" through the circuit based on all the broadcasted input signal values and the wire masks determined in the execution of Π_{Garble}, and then fix the wire mask for the final circuit output wire w to be $e_w - v_w$, where v_w is the output from $\mathcal{F}_{\mathsf{AC}}$. (If there are multiple output wires then do similarly for each.)
5. For all honest parties, for all gates and wires, sample new wire keys uniformly at random.
6. Fix the shares of honest parties so that the gates are consistent with keys and ciphertexts from Step 5 instead of the ones generated in the honest execution of Π_{Garble} in Step 1. Specifically, for each honest party P_i, for every $j \in [n] \backslash \{i\}$, for every $\gamma \in \mathbb{F}_p$, set

$$\widetilde{g}^{j,i}_{\mathsf{g},\gamma} = F_{\mathsf{k}^i_{u,\gamma}}(g,j) + \rho^i_{j,g,\mathsf{g},\gamma}$$
$$\widetilde{g}^{j,i}_{\mathsf{e},\gamma} = F_{\mathsf{k}^i_{v,\gamma}}(g,j) + \rho^i_{j,g,\mathsf{e},\gamma}$$

   and then for every $\gamma \in \mathbb{F}_p$, set

$$\widetilde{g}^{i,i}_{\mathsf{g},\gamma} = F_{\mathsf{k}^i_{u,\gamma}}(g,i) + \mathsf{k}^i_{w,e_w} - \sum_{j \neq i} \rho^j_{i,g,\mathsf{g},\gamma}$$
$$\widetilde{g}^{i,i}_{\mathsf{e},\gamma} = F_{\mathsf{k}^i_{v,\gamma}}(g,i) - e_u \mathsf{k}^i_{v,e_v} - \sum_{j \neq i} \rho^j_{i,g,\mathsf{e},\gamma}$$

   (Note that $\mathcal{S}$ knows the values of $\rho^j_{i,g,\mathsf{e},e_v}$ for all $(i,j) \in [n]^2$ because it emulates the copies of $\mathcal{F}_{\mathsf{COPE}}$ locally, from which these values are computed.)
7. Open the gates honestly by sending the shares of (emulated) honest parties to corrupt parties.
8. Await the call to $\mathcal{F}_{\mathsf{MPC}}$ to open the final output mask(s) and execute this honestly. If an emulated honest party would abort, send Abort to $\mathcal{F}_{\mathsf{AC}}$, and otherwise send OK.

The execution of Π_{Garble} is simulated perfectly by the simulator, which samples contributions to masks and keys for emulated honest parties. The fact that the simulator samples inputs on behalf of emulated honest parties in Step 2 does not affect the correctness of simulation or change the distribution as viewed by the environment in any way, because each broadcasted external wire value is a

real value masked by a uniform wire mask not revealed to the environment, and the final output is fixed by the simulator to the correct value regardless of these sampled values and the path traversed through the garbled circuit. In Step 6, the simulator fixes shares as honest parties would for the contributions to ciphertexts encrypting keys generated by corrupt parties, but for honest parties it fixes the shares so that the parties will compute the uniformly-sampled keys from Step 5 instead of keys according to a global difference that was chosen in Step 1. The reason for fixing keys and ciphertexts for honest parties in this way is that the honest party's actual global difference will be different from the simulator's sampled random global difference (used in the simulation of $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$) with high probability, so instead of arguing that a set of ciphertexts generated using keys with a fixed global difference is indistinguishable from another set of ciphertexts generated using keys with a *different* global difference (which is an alternative way to define the simulator), we show that such a set of ciphertexts is indistinguishable from a set of ciphertexts generated using uniformly-randomly sampled keys. This latter assumption is exactly the correlation-robustness assumption. Furthermore, since this is the *only* point in the distribution as viewed by the environment that is potentially different from a real execution, this is the *only* way to distinguish. Thus any environment that can distinguish between the hybrid and ideal worlds must do so by observing a difference in the distributions of ciphertexts generated, which breaks the PRF assumption.

It remains to show how to use a distinguishing environment $\mathcal{Z}$ to construct an adversary $\mathcal{D}$ (for the game outlined above). To this end, we first define a modified simulator $\mathcal{S}'$ that uses the oracle $\mathcal{O}$ provided to $\mathcal{D}$. The simulator $\mathcal{S}'$ executes exactly as $\mathcal{S}$, but it fixes an arbitrary choice of honest party P_{i^*}, and when it fixes the keys and ciphertexts in Step 6, it alters the shares of P_{i^*} for each ciphertext as follows:

For every $j \in [n] \setminus \{i^*\}$, for every $\gamma \in \mathbb{F}_p \setminus \{0\}$, set

$$\widetilde{g}^{j,i^*}_{\mathsf{g},e_u+\gamma} = \mathcal{O}\left(\mathsf{k}^{i^*}_{u,e_u}, \gamma, (g,j), 0\right) + \rho^{i^*}_{j,g,\mathsf{g},e_u+\gamma}$$
$$\widetilde{g}^{j,i^*}_{\mathsf{e},e_v+\gamma} = \mathcal{O}\left(\mathsf{k}^{i^*}_{v,e_v}, \gamma, (g,j), 0\right) + \rho^{i^*}_{j,g,\mathsf{e},e_v+\gamma}$$

and set

$$\widetilde{g}^{j,i^*}_{\mathsf{g},e_u} = F_{\mathsf{k}^{i^*}_{u,e_u}}(g,j) + \rho^{i^*}_{j,g,\mathsf{g},e_u}$$
$$\widetilde{g}^{j,i^*}_{\mathsf{e},e_v} = F_{\mathsf{k}^{i^*}_{v,e_v}}(g,j) + \rho^{i^*}_{j,g,\mathsf{e},e_v}.$$

Then set

$$\widetilde{g}^{i^*,i^*}_{\mathsf{g},e_u+\gamma} = \mathcal{O}\left(\mathsf{k}^{i^*}_{u,e_u}, \gamma, (g,i^*), 0\right) + \mathsf{k}^{i^*}_{w,e_w} - \sum_{j\neq i^*} \rho^j_{i^*,g,\mathsf{g},e_u}$$
$$\widetilde{g}^{i^*,i^*}_{\mathsf{e},e_v+\gamma} = \mathcal{O}\left(\mathsf{k}^{i^*}_{v,e_v}, \gamma, (g,i^*), 0\right) - e_u\mathsf{k}^{i^*}_{v,e_v} - \sum_{j\neq i^*} \rho^j_{i^*,g,\mathsf{e},e_v}$$

and set

$$\widetilde{g}^{i^*,i^*}_{\mathsf{g},e_u} = F_{\mathsf{k}^{i^*}_{u,e_u}}(g,i^*) + \mathsf{k}^{i^*}_{w,e_w} - \sum_{j\neq i^*} \rho^j_{i^*,g,\mathsf{g},e_u}$$
$$\widetilde{g}^{i^*,i^*}_{\mathsf{e},e_v} = F_{\mathsf{k}^{i^*}_{v,e_v}}(g,i^*) - e_u\mathsf{k}^{i^*}_{v,e_v} - \sum_{j\neq i^*} \rho^j_{i^*,g,\mathsf{e},e_v}.$$

Now observe that all queries to the oracle $\mathcal{O}$ are legal because:
- In all queries, $\gamma \neq 0$.
- Although the fourth entry is 0 for all queries, the remainder of the query message is different in every query.

Thus both requirements on the oracle queries are met.

We now show that the execution of $\mathcal{S}'$ with $\mathcal{Z}$ is the same as hybrid-world and ideal-world executions with $\mathcal{S}$.

Claim. The execution of $\mathcal{Z}$ with $\mathcal{S}'$ when $\mathcal{O} = \mathcal{F}_{\mathsf{RO}}$ is indistinguishable from the execution of $\mathcal{Z}$ with $\mathcal{S}$ in the ideal world.

Proof. If the oracle is $\mathcal{O} = \mathcal{F}_{\mathsf{RO}}$, then in the execution with $\mathcal{S}'$, every ciphertext indexed by i^* is a one-time-pad encryption. All the keys in $\mathcal{S}$ are uniformly sampled (i.e., they are not generated using a global random difference), so if there is a distinguisher between the distribution of each set of p ciphertexts under the keys $\{\mathsf{k}^{i^*}_{w,\gamma}\}_{\gamma\in p}$ and the uniform distribution, then there is a distinguisher for the PRF. Since this does not exist by assumption, the claim follows. ■

Claim. The execution of $\mathcal{Z}$ with $\mathcal{S}'$ when $\mathcal{O} = \mathcal{O}_{F,R}$ is indistinguishable from an execution in the $\mathcal{F}_{\mathsf{Commit}}$,$\mathcal{F}_{\mathsf{COPE}}$,$\mathcal{F}_{\mathsf{MPC}}$,$\mathcal{F}_{\mathsf{Rand}}$-hybrid world.

Proof. It is easy to verify that if the oracle is $\mathcal{O} = \mathcal{O}_{F,R}$, then the ciphertexts generated according to the simulation with $\mathcal{S}'$ follow exactly the distribution as in a $\mathcal{F}_{\mathsf{Commit}}$,$\mathcal{F}_{\mathsf{COPE}}$,$\mathcal{F}_{\mathsf{MPC}}$,$\mathcal{F}_{\mathsf{Rand}}$-hybrid-world execution, where the honest party P_{i^*} has global difference R that is the same as the fixed value R in the oracle's definition, where effectively the key $\mathsf{k}^{i^*}_{u,e_u}$ is sampled instead of the key $\mathsf{k}^{i^*}_{u,0}$, in order to make the oracle queries legal (which makes no difference to the distribution of the resulting keys). ■

Now we define the distinguisher $\mathcal{D}$ to execute the environment against the simulator $\mathcal{S}'$ using the oracle $\mathcal{O}$ provided the challenger of its game. If the environment guesses the execution was the ideal world, then the distinguisher guesses $b' = 0$; if the environment guesses the execution was the hybrid world, then the distinguisher guesses $b' = 1$. Thus if there is an environment that can distinguish between worlds with non-negligible advantage, then the distinguisher $\mathcal{D}$ defined above wins the security game defined above that breaks the correlation-robustness assumption on the PRF, and thus the protocol UC-securely realises $\mathcal{F}_{\mathsf{AC}}$ assuming the PRF satifies correlation-robustness. ■

Subprotocol Π_{Garble}

For simplicity, the session identifiers for functionalities are taken as implicit.

Initialise

1. Agree on a new session identifer, a computational and statistical security parameter, κ and σ, and a circuit C to evaluate, with circuit input wires W_{IN}, circuit output wires W_{OUT}, and a set of gates G comprised of a set of multiplication gates G_{MUL}, a set of addition gates G_{ADD}, and a set of selection gates G_{SEL}. Let $\texttt{PID}() : W_{\text{IN}} \to [n]$ denote the map determining which party provides input on which wire.
2. Set $\ell_\kappa := \lceil \kappa / \log p \rceil$.
3. For each $i \in [n]$, P_i samples $R^i \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and then the parties execute the procedure **Initialise** from $\Pi_{\mathsf{Mask} \times \mathsf{Diff}}$.
4. Call an instance of $\mathcal{F}_{\mathsf{MPC}}$ with input $(\mathsf{Initialise}, \mathbb{F}_p, \mathsf{sid})$.

Wire Masks and Keys

Circuit Input Wires For circuit input wire $w \in W_{\text{IN}}$, let $i := \texttt{PID}(w)$ and then do the following:

1. Party P_i samples $\lambda_w \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ with this value as input.
2. Each party P_j, $j \in [n]$, samples a key $\mathsf{k}^j_{w,0} \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and for each $\alpha \in \mathbb{F}_p$ sets $\mathsf{k}^j_{w,\alpha} := \mathsf{k}^j_{w,0} + \alpha \cdot R^j$.

Addition Output Wires For each wire w that is an output of an addition gate with input wires u and v, do the following:

1. Compute $[\![\lambda_w]\!] = [\![\lambda_u + \lambda_v]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
2. For each $i \in [n]$, party P_i computes $\mathsf{k}^i_{w,0} := \mathsf{k}^i_{u,0} + \mathsf{k}^i_{v,0}$ and for each $\alpha \in \mathbb{F}_p$ sets $\mathsf{k}^i_{w,\alpha} := \mathsf{k}^i_{w,0} + \alpha \cdot R^i$.

Multiplication Output Wires For a wire w that is an output of a multiplication gate with input wires u and v,

1. For each $\mathsf{x} \in \{\mathsf{g}, \mathsf{e}\}$,
   (a) For each $i \in [n]$, party P_i samples $\lambda^i_{w,\mathsf{x}} \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ with this value as input.
   (b) Compute $[\![\lambda_{w,\mathsf{x}}]\!] := [\![\sum_{i=1}^n \lambda^i_{w,\mathsf{x}}]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
   (c) For each $i \in [n]$, party P_i samples a key $\mathsf{k}^i_{w,\mathsf{x},0} \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and for each $\gamma \in \mathbb{F}_p$ sets $\mathsf{k}^i_{w,\mathsf{x},\gamma} := \mathsf{k}^i_{w,\mathsf{x},0} + \gamma \cdot R^i$.

2. For each $i \in [n]$, party P_i sets $\mathsf{k}^i_{w,0} := \mathsf{k}^i_{w,\mathsf{g},0} + \mathsf{k}^i_{w,\mathsf{e},0}$ and for all $\gamma \in \mathbb{F}_p$ sets $\mathsf{k}^i_{w,\gamma} := \mathsf{k}^i_{w,0} + \gamma \cdot R^i$.

Wire Mask/Global Difference Products

Multiplication Gates For each $g \in G_{\text{MUL}}$, let u and v be the input wires and w the output wire; then do the following:

1. Compute $[\![\lambda_{uv}]\!] := [\![\lambda_u \cdot \lambda_v]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
2. Execute the procedure **Multiply** from $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ on the set $\{\lambda_u, \lambda_v, \lambda_{uv}, \lambda_{w,\mathsf{g}}, \lambda_{w,\mathsf{e}}\}_{g\in G_{\text{MUL}}}$ to obtain, for all $i \in [n]$, (unauthenticated) sharings

$$\left\{\langle R^i \cdot \lambda_u\rangle, \langle R^i \cdot \lambda_v\rangle, \langle R^i \cdot \lambda_{uv}\rangle, \langle R^i \cdot \lambda_{\mathsf{g},w}\rangle, \langle R^i \cdot \lambda_{\mathsf{e},w}\rangle\right\}_{g\in G_{\text{MUL}}}.$$

3. For each $i \in [n]$, for each $\gamma \in \mathbb{F}_p$, set

$$\langle \rho_{i,g,\mathsf{g},\gamma}\rangle := -\gamma \cdot \langle R^i \cdot \lambda_v\rangle + \langle R^i \cdot \lambda_{uv}\rangle + \langle R^i \cdot \lambda_{\mathsf{g},w}\rangle$$
$$\langle \rho_{i,g,\mathsf{e},\gamma}\rangle := -\gamma \cdot \langle R^i \cdot \lambda_u\rangle + \langle R^i \cdot \lambda_{\mathsf{e},w}\rangle$$

Garbling

Multiplication Gates For each $g \in G_{\text{MUL}}$, for each $i \in [n]$, for each $\gamma \in \mathbb{F}_p$,

1. The parties compute the garbler half gate:
   - P_i sets $\widetilde{g}^{i,i}_{\mathsf{g},\gamma} := F_{\mathsf{k}^i_{u,\gamma}}(g,i) + \mathsf{k}^i_{w,\mathsf{g},0} + \rho^i_{i,g,\mathsf{g},\gamma}$
   - For every $j \neq i$, P_j sets $\widetilde{g}^{i,j}_{\mathsf{g},\gamma} := F_{\mathsf{k}^j_{u,\gamma}}(g,i) + \rho^j_{i,g,\mathsf{g},\gamma}$
2. The parties compute the evaluator half gate:
   - Party P_i sets $\widetilde{g}^{i,i}_{\mathsf{e},\gamma} := F_{\mathsf{k}^i_{v,\gamma}}(g,i) + \mathsf{k}^i_{w,\mathsf{e},0} - \gamma \cdot \mathsf{k}^i_{u,0} + \rho^i_{i,g,\mathsf{e},\gamma}$
   - Every party P_j, $j \neq i$, sets $\widetilde{g}^{i,j}_{\mathsf{e},\gamma} := F_{\mathsf{k}^j_{v,\gamma}}(g,i) + \rho^j_{i,g,\mathsf{e},\gamma}$

Fig. 7: Subprotocol Π_{Garble} for garbling a circuit.

Subprotocol Π_{Eval}

Input Wires For each wire $w \in W$ which is an input wire, the parties do the following:

1. Let $i = \mathtt{PID}(w)$: then party P_i computes and broadcasts $e_w := v_w + \lambda_w$, where $v_w \in \mathbb{F}_p$, is P_i's input.
2. For each $i \in [n]$, party P_i broadcasts k^i_{w,e_w}.

Opening For each $g \in G_{\text{MUL}}$, for each $\mathsf{x} \in \{\mathsf{g},\mathsf{e}\}$, for each $i \in [n]$,

1. For each $j \in [n]$, for each $\gamma \in \mathbb{F}_p$, P_i broadcasts $\widetilde{g}^{j,i}_{\mathsf{x},\gamma}$.
2. All parties compute $\widetilde{g}^i_{\mathsf{x},\gamma} := \sum_{j=1}^n \widetilde{g}^{i,j}_{\mathsf{x},\gamma}$.

Circuit Evaluation Traversing the circuit in topological order, for every gate G with input wires u and v and output wire w, the parties do the following:

- If g is an addition gate, each party does the following:
  1. Set the external wire value to be $e_w := e_u + e_v$.

2. Compute the output keys as: for each $i \in [n]$, $\mathsf{k}^i_{w,e_w} := \mathsf{k}^i_{u,e_u} + \mathsf{k}^i_{v,e_v}$.

– If g is a multiplication gate, each party does the following:

1. For each $i \in [n]$, compute

$$\mathsf{k}^i_{w,e_w} := \underbrace{\widetilde{g}^i_{\mathsf{g},e_u} - \sum_{j=1}^{n} F_{\mathsf{k}^j_{u,\mathsf{g},e_u}}(g,i)}_{\text{Garbler half gate}} + \underbrace{\widetilde{g}^i_{\mathsf{e},e_v} - \sum_{j=1}^{n} F_{\mathsf{k}^j_{v,\mathsf{e},e_v}}(g,i)}_{\text{Evaluator half gate}} + e_v \cdot \mathsf{k}^i_{u,e_u}.$$

2. Each party P_i determines the signal value e_w by computing $e_w := (\mathsf{k}^i_{w,e_w} - \mathsf{k}^i_{w,0}) \cdot (R^i)^{-1}$.

Output To obtain the output of wire $w \in W_{\text{OUT}}$, call $\mathcal{F}_{\mathsf{MPC}}$ to execute the procedure **Output** to reveal the value $e_w - [\![\lambda_w]\!]$.

Fig. 8: Subprotocol Π_{Eval} for evaluating the garbled circuit.

All of the protocols in this section can be realised using protocols (with minor modifications) given in MASCOT [16]; however, the two parts of the computation outlined above are most optimally performed using a mixed approach: using the Overdrive protocol [17] to realise $\mathcal{F}_{\mathsf{MPC}}$, and using MASCOT-like protocols to perform the Wire Mask/Global Difference products. The reason is that Overdrive is more efficient over large prime fields, as opposed to large extension fields such as $\mathbb{F}_{2^k}$ for which MASCOT is better.

4 Selector Gate

It was argued by Ben-Efraim [6] that a selector gate taking a Boolean selection bit and choosing between field elements is a desirable feature of garbling protocols as the selection bit is likely to come from the evaluation of some Boolean subcircuit. Such a construction was given in [6]; in this section we give an alternative construction, which we call the alternative selector gate, which, specifically, takes one input in $\mathbb{F}_2$, held as a signal bit with a corresponding key in $\mathbb{F}_{2^\kappa}$ and viewed as output from a Boolean circuit, and two inputs in $\mathbb{F}_p$, and outputs one of the field elements according to the selection bit. Note that if the selection bit is also a field element then the standard $2 \cdot p$ ciphertexts for general field/field multiplication is required, as is the case in Ben-Efraim's work [6].

Multifield Shared Bits Rotaru and Wood [24] showed how to generate secret-sharings of uniformly-random bits shared in two fields with authentication in each; these were called daBits, for doubly-authenticated bits. This can be viewed as an actively-secure version of the multi-field bits discussed by Ben-Efraim, which can be used in arithmetic garbling of selector gates. The protocol for

generating such bits uses authentication in a black-box way, and so any actively-secure MPC protocol can be used to generate them. In this work, we use daBits shared in $\mathbb{F}_p$ and $\mathbb{F}_{2^\kappa}$ for our selector gates.

4.1 New Selector Gate

Recall that the standard cost of multiplication in $\mathbb{F}_p$ is $p \cdot p$ ciphertexts; the garbler/evaluator half-gate approach reduces this to $p+p$ ciphertexts. The main observation driving our alternative selector gate is that the actual selection operation is a multiplication of a bit by an element in $\mathbb{F}_p$, and thus the goal is to reduce the naïve $2 \cdot p$ ciphertexts to (almost) $2 + p$.

A selection gate based on selection bit b between the values on wires u and v is computed via the standard multiplexer $u + (v - u) \cdot b$. Since linear operations are garbled without communication or preprocessing, we focus on the product of the wire $w := v - u \in \mathbb{F}_p$ with the bit $b \in \mathbb{F}_2$; the output wire is denoted by z.

The point is that while the previous approach by Ben-Efraim involved converting the bit to $\mathbb{F}_p$ using a so-called *projection gate* and evaluating a standard multiplication gate in $\mathbb{F}_p$, we can use daBits to perform this projection directly. We will now explain how to garble the new selector gate; this explanation is followed by a formal protocol description.

Let b' be the Boolean wire, and let b be the $\mathbb{F}_p$ wire to which we wish to convert. We let the wire mask output of the Boolean wire be a daBit $\lambda_{b'} \in \{0, 1\}$ and convert it to an $\mathbb{F}_p$ wire using $2n$ ciphertexts in $\mathbb{F}_{p^{\ell_\kappa}}$ as follows: for every $\beta \in \{0, 1\}$, for every $j \in [n]$,

$$g^j_\beta := \sum_{i=1}^{n} F_{\mathsf{k}^i_{b',\beta}}(g, j) + \mathsf{k}^j_{b,0} + ((\beta + \lambda_{b'} - 2 \cdot \beta \cdot \lambda_{b'}) + \lambda_b) \cdot R^j,$$

where $\mathsf{k}^i_{b',\beta} \in \mathbb{F}_{2^\kappa}$ for all $i \in [n]$ and λ_b is a uniform mask in $\mathbb{F}_p$. Since the PRF used in previous sections takes keys of length at least κ bits, we may assume the same PRF is used here, with additional padding if necessary. Here, we use the fact that in any field, if a and b are in $\{0, 1\}$ then their XOR is computed as

$$a \oplus b = a + b - 2 \cdot a \cdot b,$$

which means that we can remove the mask in $\mathbb{F}_p$ since the mask $\lambda_{b'}$ used in the garbling of the Boolean circuit was a daBit. The two ciphertexts (for each $i \in [n]$) are indexed by the two possible Boolean external values, which is denoted by $e_{b'}$; the external value on the output, denoted by e_b, is not needed in the next steps, but can be computed by the evaluators in the usual way (i.e., by P_i comparing the output key indexed by i to its own p keys). In doing so, the evaluators learn

either $0 + \lambda_b$ or $1 + \lambda_b$, but do not learn which they hold. In fact, this external value e_b is never used by the evaluators.

The multiplication gate is then computed in two halves:

$$g^j_{\mathsf{g},\alpha} := \sum_{i=1}^{n} F_{\mathsf{k}^i_{w,\alpha}}(g,j) + \mathsf{k}^j_{\mathsf{g},z,0} - \alpha(\mathsf{k}^j_{b,0} + \lambda_b R^j)$$

$$g^j_{\mathsf{e},\beta} := \sum_{i=1}^{n} F_{\mathsf{k}^i_{b,\beta}}(g,j) + \mathsf{k}^j_{\mathsf{e},z,0} - (\beta + \lambda_{b'} - 2\beta\lambda_{b'})\lambda_w R^j + \lambda_z R^j$$

Now when evaluating, the parties will obtain e_w and $e_{b'}$, will compute $a := \mathsf{Dec}(g_{\mathsf{g},e_w})$ and $b := \mathsf{Dec}(g_{\mathsf{e},e_{b'}})$ and will compute

$$\begin{aligned}
\mathsf{k}_{z,e_z} &= a + b + e_w \mathsf{k}_{b,e_b} \\
&= \left(\mathsf{k}^j_{\mathsf{g},z,0} - e_w(\cancel{\mathsf{k}^j_{b,0}} + \lambda_b R^j)\right) + \left(\mathsf{k}^j_{\mathsf{e},z,0} - (e_{b'} + \lambda_{b'} - 2e_{b'}\lambda_{b'})\lambda_w R^j + \lambda_z R^j\right) \\
&\qquad + e_w(\cancel{\mathsf{k}_{b,0}} + e_b R^j) \\
&= \left(\mathsf{k}^j_{\mathsf{g},z,0} - e_w\lambda_b R^j\right) + \left(\mathsf{k}^j_{\mathsf{e},z,0} - v_b\lambda_w R^j + \lambda_z R^j\right) + e_w e_b R^j \\
&= \left(\mathsf{k}^j_{\mathsf{g},z,0} - (v_w + \lambda_w)\lambda_b R^j\right) + \left(\mathsf{k}^j_{\mathsf{e},z,0} - v_b\lambda_w R^j + \lambda_z R^j\right) \\
&\qquad + (v_w + \lambda_w)(v_b + \lambda_b)R^j \\
&= \left(\mathsf{k}^j_{\mathsf{g},z,0} - \cancel{(v_w + \lambda_w)\lambda_b R^j}\right) + \left(\mathsf{k}^j_{\mathsf{e},z,0} - v_b\lambda_w R^j + \lambda_z R^j\right) \\
&\qquad + ((v_w + \lambda_w)v_b + \cancel{(v_w + \lambda_w)\lambda_b})R^j \\
&= \mathsf{k}^j_{\mathsf{g},z,0} + \left(\mathsf{k}^j_{\mathsf{e},z,0} + \lambda_z R^j - \cancel{v_b\lambda_w R^j}\right) + ((v_w + \cancel{\lambda_w})v_b)R^j \\
&= \mathsf{k}^j_{z,0} + (v_w v_b + \lambda_z)R^j.
\end{aligned}$$

In total, this requires $p + 4$ ciphertexts per party: 2 for the conversion, 2 for the first half gate and p for the second.

We do not provide a complete proof of the security of the alternative selector gate as it follows straightforwardly from the security of the selector gate of Ben-Efraim [6]. The high-level intuition is that the keys $\mathsf{k}^j_{\mathsf{g},z,0}$ and $\mathsf{k}^j_{\mathsf{e},z,0}$ are sampled uniformly at random, and independently of one another, and so their sum $\mathsf{k}_{z,0}$ is also uniformly random, as is required of 0 keys; furthermore, the wire mask λ_z is uniform and not known to any individual party, so the external value of the output wire perfectly hides the real value $v_w \cdot v_b$. The complete protocol for garbling these new selector gates is given in Figure 9. The evaluation protocol is the same as the evaluation of a multiplication gate and is therefore omitted.

Subprotocol Π_{Select}

This subprotocol takes a gate with Boolean input wire b' and arithmetic inputs u and v and output wire $z = u + (v - u) \cdot b'$.

Wire Masks and Keys

Wire Mask/Global Difference Products

Selection Gates If g is a selection gate with input wires u and v, selection bit wire b, and output wire z,

1. If b' is the Boolean input wire, let $\lambda_{b'}$ be the $\mathbb{F}_p$ daBit mask stored as $[\![\lambda'_b]\!]$ in $\mathcal{F}_{\mathsf{MPC}}$.
2. Generate an $\mathbb{F}_p$ wire mask $[\![\lambda_b]\!]$:
   (a) For each $i \in [n]$, party P_i samples $\lambda_b^i \xleftarrow{\$} \mathbb{F}_p$ and calls $\mathcal{F}_{\mathsf{MPC}}$ with this value as input.
   (b) Compute $[\![\lambda_b]\!] := [\![\sum_{i=1}^n \lambda_b^i]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
3. Each party P_i samples a key $\mathsf{k}_{b,0}^i \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$.
4. Generate an $\mathbb{F}_p$ output wire mask $[\![\lambda_z]\!]$ in the same way as for $[\![\lambda_b]\!]$, above.
5. Let $[\![\lambda_u]\!]$ and $[\![\lambda_v]\!]$ be the masks stored in $\mathcal{F}_{\mathsf{MPC}}$ for wires u and v, respectively, generated when garbling an addition or multiplication gate or input wire. Set $[\![\lambda_w]\!] := [\![\lambda_v]\!] - [\![\lambda_u]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
6. Let $\mathsf{k}_{u,0}^i$ and $\mathsf{k}_{v,0}^i$ be the keys previously generated by party P_i for wires u and v. For each $i \in [n]$, party P_i sets $\mathsf{k}_{w,0}^i := \mathsf{k}_{v,0}^i - \mathsf{k}_{u,0}^i$.
7. For each $i \in [n]$, party P_i samples a wire key $\mathsf{k}_{w,\mathsf{g},0}^i \xleftarrow{\$} \mathbb{F}_{p^{\ell_\kappa}}$ and sets $\mathsf{k}_{w,\mathsf{e},0}^i := \mathsf{k}_{w,0}^i - \mathsf{k}_{w,\mathsf{g},0}^i$.
8. Compute $[\![\lambda_{b'w}]\!] := [\![\lambda_{b'} \cdot \lambda_w]\!]$ by calling $\mathcal{F}_{\mathsf{MPC}}$.
9. Execute $\Pi_{\mathsf{Mask}\times\mathsf{Diff}}$ to obtain
$$\Big\{\langle R^i \cdot \lambda_{b'}\rangle, \langle R^i \cdot \lambda_b\rangle, \langle R^i \cdot \lambda_w\rangle, \langle R^i \cdot \lambda_{b'w}\rangle, \langle R^i \cdot \lambda_z\rangle\Big\}.$$
10. For each $i \in [n]$, for each $\alpha \in \mathbb{F}_p$ and $\beta \in \{0,1\}$, set
$$\begin{aligned}
\langle \rho_{i,g,b,\beta}\rangle &:= (1 - 2\cdot\beta)\cdot\langle R^j \cdot \lambda_{b'}\rangle + \langle R^j \cdot \lambda_b\rangle \\
\langle \rho_{i,g,\mathsf{g},\alpha}\rangle &:= -\alpha \cdot \langle R^j \cdot \lambda_b\rangle \\
\langle \rho_{i,g,\mathsf{e},\beta}\rangle &:= -\beta \cdot \langle R^j \cdot \lambda_w\rangle - (1 - 2\cdot\beta)\cdot\langle R^j \cdot \lambda_{b'w}\rangle + \langle R^j \cdot \lambda_z\rangle.
\end{aligned}$$

Garbling

Selection Gates If g is a selection gate with input wires u and v and selection bit wire b,

1. The parties generate ciphertexts for converting the Boolean input wire b' to an $\mathbb{F}_p$ wire b: for every $i \in [n]$,
   – P_i sets $\widetilde{g}_{b,\beta}^{i,i} := F_{\mathsf{k}_{b,\beta}^i}(g,i) + \mathsf{k}_{b,0}^i + \beta \cdot R^i + \rho_{i,g,b,\beta}^i$
   – For every $j \neq i$, P_j sets $\widetilde{g}_{b,\beta}^{i,j} := F_{\mathsf{k}_{b,\beta}^i}(g,i) + \rho_{i,g,b,\beta}^j$
2. The parties compute the gates for the product of wire $w := (v - u)$ with wire b:

(a) The parties compute the garbler half gate:
- P_i sets $\widetilde{g}^{i,i}_{\mathsf{g},\alpha} := F_{\mathsf{k}^i_{\mathsf{g},z,\alpha}}(g,i) + \mathsf{k}^i_{\mathsf{g},z,0} - \alpha \cdot \mathsf{k}^i_{b,0} + \rho^i_{i,g,\mathsf{g},\alpha}$
- For every $j \neq i$, P_j sets $\widetilde{g}^{i,j}_{\mathsf{g},\alpha} := F_{\mathsf{k}^j_{\mathsf{g},z,\alpha}}(g,i) + \rho^j_{i,g,\mathsf{g},\alpha}$

(b) The parties compute the evaluator half gate:
- P_i sets $\widetilde{g}^{i,i}_{\mathsf{e},\beta} := F_{\mathsf{k}^i_{\mathsf{e},b,\beta}}(g,i) + \mathsf{k}^i_{\mathsf{e},z,0} + \rho^i_{i,g,\mathsf{e},\beta}$
- For every $j \neq i$, P_j sets $\widetilde{g}^{i,j}_{\mathsf{e},\beta} := F_{\mathsf{k}^j_{\mathsf{e},b,\beta}}(g,i) + \rho^j_{i,g,\mathsf{e},\beta}$

Fig. 9: Subprotocol Π_{Select} for garbling a selector gate.

5 Evaluation in comparison to previous work

We evaluate our work in comparison to all previous works in the field of arithmetic garbling; both in the two-party, and in the multiparty paradigm. As shown in Table 1, we are the only work providing full-threshold active security, and proving our garbling techniques UC-secure under the named assumptions. Previous work provided either two-party, passively secure constructions [3, 4], or multiparty, passively secure constructions in the honest majority setting [6].

Recall that, in the multiparty setting, projection gates (significantly increasing the efficiency of previous work [3, 4]) are non-trivial to construct, and are not universal (i.e., tailored techniques per gate are required). Given that in the multiparty setting all parties play the role of the garbler, we cannot exploit the asymmetry between garbler and evaluator that two-party solutions enjoy. In addition, as already pointed out by Ben-Efraim [6], each garbled table row in the multiparty setting requires n ciphertexts, versus a single ciphertext in the two-party setting, and each row decryption requires n^2 PRF calls in the multiparty setting, versus a single PRF call in the two-party setting. These values are reflected in our cost description provided in Table 1.

For the works that did not suggest an improved version of a specific garbled gate (e.g., multiplication gates in both our work, and the work of Ball et al. [3]), we assume the same cost as the cost of the best previous technique of which they make use. Our work almost halves the communication cost of the selector gate, compared to the previous work in the multiparty setting [6], at the cost of losing the $\sim 33\%$ improvement of computation cost that Ben-Efraim's approach enjoys (in addition to the generation of daBits). This is an overall improvement, given that the main bottleneck is the communication cost, and that the computation cost is dominated by hash function calls, which are efficient. Garbling is a technique suitable for secure computation over unreliable networks, where continuous connectivity cannot be guaranteed. Although most of the communication happens during the preprocessing phase, the communication cost remains

the main bottleneck of garbling. Performing one additional PRF call during the online phase, given that it comes at such a significant efficiency increase of the offline phase, is less of a concern, since PRFs are a symmetric primitive, with significant hardware optimisations on modern processors. Our selector gate remains competitive even with the related work in the two party setting [3], where we consider a selector gate to be the so-called cross-modulus multiplication for $q = 2$. We require $p + 4$ ciphertexts per party, while Ball et al. [3] require $p + 1$. This minor difference comes mainly from the fact that we cannot deploy the row reduction techniques in the multiparty setting. Note that the cost of our protocols that comes from applying actively secure MPC techniques, instead of the passively secure approach in previous works, is not digested in Table 1.

Protocol	Model	Parties	Multiplication		Selection	
			#Ciphertexts	#Decryptions	#Ciphertexts	#Decryptions
[4]	passive	2	$6p - 5$	6	$2p - 1$	2
[6]	passive	n	$2p \cdot n$	$2n^2$	$(2p + 2) \cdot n$	$2n^2$
[3]	passive	2	$6p - 5$	6	$p + 1$	2
Ours	active	n	$2p \cdot n$	$2n^2$	$(p + 4) \cdot n$	$3n^2$

Table 1: Comparison of our garbling techniques with the garbling of [4], [6], and [3], in terms of security model supported, number of parties supported, number of ciphertexts required per multiplication and selection gate, and number of decryptions required per multiplication and selection gate.

6 Conclusion

Our work continues the study of multiparty arithmetic garbling initiated by Ben-Efraim [6]. Specifically, we extend the previous work from the semi-honest, honest majority setting, to the full-threshold actively-secure setting. Given the practical importance of circuits, which combine Boolean and arithmetic gates, we follow this paradigm, also considered in the work of Ben-Efraim [6]. We consider a selector gate as suggested by Ben-Efraim [6] (essentially a multiplexer); we extend it to the full-threshold actively-secure equivalent, and show how to garble such a gate, while almost halving the communication cost it incurs.

Representations of Boolean circuits have clear advantages over arithmetic circuits when it comes to non-linear operations. On the other hand, appropriate representations of arithmetic circuits are orders of magnitude more efficient than Boolean circuits for linear operations on arithmetic values. Garbling techniques that enable the construction of circuits, which integrate both Boolean and arithmetic gates, are essential to treat numerous real-world application scenarios, and

allow computation of arbitrary circuits in constant rounds. This is the reason why the design of such garbling schemes is on the rise. It remains an interesting open problem to devise techniques that allow a seamless and efficient conversion between the two representations with active security in the multiparty setting.

References

1. Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P Smart, and Tim Wood. Zaphod: Efficiently combining lsss and garbled circuits in scale. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 33–44, 2019.
2. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.
3. Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. Garbled neural networks are practical. Cryptology ePrint Archive, Report 2019/338, 2019. https://eprint.iacr.org/2019/338.
4. Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 565–577, Vienna, Austria, October 24–28, 2016. ACM Press.
5. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.
6. Aner Ben-Efraim. On multiparty garbling of arithmetic circuits. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 3–33, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.
7. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 578–590, Vienna, Austria, October 24–28, 2016. ACM Press.
8. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.
9. Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High performance multiparty computation for binary circuits based on oblivious transfer. Cryptology ePrint Archive, Report 2015/472, 2015. http://eprint.iacr.org/2015/472.

10. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/2000/067.

11. Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 769–798, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

12. 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, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

13. Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 629–659, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

14. Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 598–628, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

15. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.

17. 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, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

18. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer, Heidelberg, Germany.

19. Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. 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 319–338, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

20. Tal Malkin, Valerio Pastro, and abhi shelat. An algebraic approach to garbling. *Unpublished manuscript*, 2016.

21. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *1st ACM Conference on Electronic Commerce, EC '99*, pages 129–139. Citeseer, 1999.

22. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

23. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany.

24. Dragos Rotaru and Tim Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. Cryptology ePrint Archive, Report 2019/207, 2019. https://eprint.iacr.org/2019/207.

25. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 39–56, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

26. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

27. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

Chapter 11

The return of Eratosthenes: Secure Generation of RSA Moduli using Distributed Sieving

Publication data

C. Delpech de Saint Guilhem, E. Makri, D. Rotaru, T. Tanguy. "The return of Eratosthenes: Secure Generation of RSA Moduli using Distributed Sieving". In *IACR Cryptology ePrint Archive*, 2021, 565.

The return of Eratosthenes: Secure Generation of RSA Moduli using Distributed Sieving

Cyprien Delpech de Saint Guilhem[1], Eleftheria Makri[1,2], Dragos Rotaru[1,3], and Titouan Tanguy[1]

[1] imec-COSIC, KU Leuven, Belgium
[2] ABRR, Saxion University of Applied Sciences, The Netherlands
[3] Cape Privacy

Abstract. Secure multiparty generation of an RSA biprime is a challenging task, which increasingly receives attention, due to the numerous privacy-preserving applications that require it. In this work, we construct a new protocol for the RSA biprime generation task, secure against a malicious adversary, who can corrupt any subset of protocol participants. Our protocol is designed for generic MPC, making it both platform-independent and allowing for weaker security models to be assumed (e.g., honest majority), should the application scenario require it. By carefully "postponing" the check of possible inconsistencies in the shares provided by malicious adversaries, we achieve noteworthy efficiency improvements. Concretely, we are able to produce additive sharings of the prime candidates, from multiplicative sharings via a semi-honest multiplication, without degrading the overall (active) security of our protocol. This is the core of our sieving technique, increasing the probability of our protocol sampling a biprime. Similarly, we perform the first biprimality test, requiring several repetitions, without checking input share consistency, and perform the more costly consistency check only in case of success of the Jacobi symbol based biprimality test. Moreover, we propose a protocol to convert an additive sharing over a ring, into an additive sharing over the integers. Besides being a necessary sub-protocol for the RSA biprime generation, this conversion protocol is of independent interest. The cost analysis of our protocol demonstrated that our approach improves the current state-of-the-art (Chen et al. – Crypto 2020), in terms of communication efficiency. Concretely, for the two-party case with malicious security, and primes of 2048 bits, our protocol improves communication by a factor of ~ 37.

1 Introduction

An *RSA modulus*, also known as a *biprime*, and usually denoted by the variable N, refers to a number which is the product of two prime numbers, usually denoted by p and q; thus, $N = p \cdot q$. The RSA modulus is a crucial component of the first public key encryption scheme, the RSA scheme [33], as well as many

other public key encryption schemes that followed it. The security of the RSA cryptosystem is based on the hardness of factoring, and as such N is part of the public key of the RSA scheme [33], while its factors p and q determine the secret key. Specifically, the security of the cryptosystem is determined by the bit-length of the biprime, and therefore efficient methods to generate (large) biprimes have been of interest since RSA was devised.

Initially, the generation of the parameters of a public key cryptosystem (including the biprime generation) was assigned to a trusted third party. However, there are applications were no single party can be entrusted with such a task, which gave rise to the study of distributed biprime generation. The problem of secure distributed RSA modulus generation is being studied since 1997, when the seminal work of Boneh and Franklin [5] appeared. After the initial interest in the problem, in the years around the work of Boneh and Franklin, with literature attempting to improve the efficiency, or security aspects of the original work, the subject ceased being studied for about a decade. Then, again, during the last decade the interest on secure distributed RSA modulus generation is increasing. This is due to the sheer number of recent applications, requiring distributed RSA modulus generation.

Traditionally, the study of distributed RSA modulus generation has found numerous applications in threshold cryptography [13–15, 19, 32]. Nowadays, blockchain applications requiring permissionless consensus, also need to deploy the techniques of threshold cryptography, which explains the recently revived interest in RSA modulus generation. A concrete example of applications for the RSA modulus generation in the context of decentralized systems and consensus protocols is that of Verifiable Delay Functions (VDFs) [4, 30, 39]. Threshold cryptography requiring the distributed generation of an RSA modulus is now expanding beyond academia, as companies and foundations (e.g., Unbound, the VDF Alliance, the Ethereum Foundation, Ligero) are providing services based on top of these technologies to the public.

1.1 Related Work

The study of secure multiparty RSA modulus generation was initiated by Boneh and Franklin [5]. Boneh and Franklin [5] devised a biprimality test to perform the distributed RSA modulus generation, instead of individually testing the primality of the two prime factors of N. Given that N is the public output of their protocol, this granted them an efficiency advantage, since their expensive multiparty computations can be computed modulo the public N this way. On the other hand, the biprimality testing approach, requires that two prime numbers are *simultaneously* sampled, which leads to an increased number of iteratively invoking the subroutines of the protocol, since efficient primality testing is probabilistic

in nature. Trial division, applied individually on each of the prime candidates, somewhat relaxes the abovementioned performance penalty. The blueprint of Boneh and Franklin [5], which is also adopted by most of the follow-up protocols in the literature, consists of three main steps: (1) pick prime candidates (via trial division); (2) securely multiply candidates; (3) biprimality testing (followed by the RSA key generation step, whenever key generation is actually needed).

The Boneh-Franklin protocol was implemented, together with some newly introduced optimizations by Malkin et al. [29]. Malkin et al. [29] first deploy a (simpler) Fermat test to check biprimality, which with low probability introduces false positives. If this simpler test passes, then they deploy the Boneh-Franklin biprimality test, to eliminate any potential false positives. The most important optimization proposed by Malkin et al. [29] is a distributed sieving technique, which results in a $10\times$ improvement in running time for the generation of a 1024-bit biprime. The distributed sieving ensures that the candidate primes p and q are not divisible by the first small primes, up to a predetermined bound. This is done by each party randomly selecting multiplicative shares, which are coprime to the predeterimined bound (and therefore their product is also coprime to the bound), and then transforming these to additive shares to proceed with the rest of the protocol.

Frankel et al. [16] were the first to propose a distributed RSA key generation in the malicious security, honest majority setting. One of the main tools, devised and used for the RSA key generation by Frankel et al. [16], is an unconditionally secure multiplication protocol over the integers. To generate the RSA modulus Frankel et al. [16] deploy a maliciously secure version of the Boneh-Franklin biprimality test, and then show how to produce the actual RSA keys: a more efficient version for small keys, and a less efficient one for larger keys. Although their tailored protocol is more efficient than a solution merely deploying a passive to active security compiler, it still remains inefficient.

Limited to the two-party case, Poupard and Stern [31] propose a maliciously secure protocol for the RSA modulus generation, based on OT. Although their protocol is less efficient than the Boneh-Franklin one, it is secure in a more stringent security model, and it can serve two parties (instead of three that are required by the Boneh-Franklin protocol), in application scenarios where this is needed. However, the protocol suffers from a leakage of information in the presence of a malicious party, who can learn up to $\sum_{p\in\mathcal{P}} \log(p)/p$ bits of the prime factor p, with $\mathcal{P}$ being the set of tested primes. Another OT-based, two-party protocol for the RSA key generation (and the RSA modulus generation) was proposed by Gilboa [20]. Unlike Poupard and Stern's solution, Gilboa's protocol offers only semi-honest security, but it is more efficient. The well-known OT-based multiplication protocol of Gilboa, which by now is a classic, and adaptations

thereof are frequently used in the construction of secure multiparty computation protocols, is also the basis of the RSA key generation protocol that they devised.

Algesheimer et al. [1] perfom a distributed primality test, unlike the protocol of Boneh and Franklin [5] that was based on biprimality testing. Specifically, they show how to perform a distributed version of the Miller-Rabin primality test, and do proceed with the multiparty computations modulo a secret prime, that the previous work avoided. To achieve this, they deploy three types of secret sharing schemes, and show how to convert shares from one to another. Moreover, their constructions allow the generation of an RSA modulus, whose prime factors are actually *safe* primes. Both the work of Algesheimer et al. [1] and the work of Boneh and Franklin [5] are proven secure in the semi-honest, honest majority security model, and require minimally three parties.

Damgård and Mikkelsen [12] were the first to efficiently achieve malicious security for the task of distributed RSA modulus generation, though in the honest majority setting. In fact, the first to adjust the protocol of Boneh and Franklin to the active security model were Frankel et al. [16], but malicious security was achieved at the cost of protocol efficiency. The work of Damgård and Mikkelsen [12] can be seen as a hybrid of two of its predecessors [1, 5], trying to combine the most efficient aspects of both of these approaches. Concretely, by using replicated secret sharing they can do both multiplications, and the modular reductions suggested by Algesheimer et al. [1], which require the protocol to work over the integers (instead of a field), without having to convert between different secret sharing schemes. The downside of the Damgård and Mikkelsen's [12] protocol is that it does not scale well in the number of parties, and it is not straightforward to extend to more than three parties.

Hazay et al. [22, 23] deploy partially homomorphic encryption to complete the two first steps of the Boneh and Franklin blueprint, and proceed to step 3 with biprimality testing. They achieve the first general n-party protocol (i.e., for any $n \geq 2$) in the active security model, and the dishonest majority setting. Active security in this setting is achieved by deploying tailored zero-knowledge proofs. Hazay et al. [22, 23] build upon Gilboa's technique [20] achieving two-party actively secure RSA modulus generation à la Boneh and Franklin, but they are the first to adapt also the trial division step of the protocol, achieving therefore a significant efficiency boost. For the multiplication step that follows to compute the candidate biprime, Hazay et al. [22,23] carefully use a combination of the Paillier and ElGamal encryption schemes.

In the two party setting, Frederiksen et al. [17] propose an OT-based, maliciously secure protocol for the distributed RSA modulus generation, which is more efficient than previous work. The efficiency improvement is due to one

single compact zero-knowledge argument of correct behavior at the end of the protocol, instead of the numerous (one per message) tailored zero-knowledge proofs required in the protocol of Hazay et al. [22, 23]. The maliciously secure protocol of Frederiksen et al. [17] is provably secure, and concretely efficient, as shown by the authors with an implementation. However, it suffers from some information leakage at the trial division step. This leakage is formalized in the functionality, and taken into account in the security proof, and it is argued to be justified both in theory and in practice, as leakage of a few bits of the prime factors should not be able to break the RSA assumption, and therefore also not the security of the protocol. Nevertheless, in the malicious case, this leakage may lead to selective failure attacks, which also impacts the efficiency of the protocol.

Recently, Chen et al. [7] successfully ovecame the limitations of the proposal of Frederiksen et al. [17]. Benefiting from the efficiency advantages that a CRT number representation allows, Chen et al. [7] devise a maliciously secure n-party protocol, tolerating $n-1$ (active) corruptions, while avoiding both the deployment of expensive cryptographic primitives, and the information leakage incurred by the protocol of Frederiksen et al. [17]. Leveraging the CRT representation, Chen et al. [7] not only gain efficiency from the linearity of the smaller in bit-size computations that can be performed locally, but they also *constructively* sample their primes such that they are not divisible by the small primes used for the CRT representation. This way they significantly increase the probability of hitting a prime, and therefore also increase the overall protocol efficiency. Their distributed RSA modulus generation protocol follows the general Boneh and Franklin paradigm.

A large scale implementation of the distributed RSA modulus generation, and improvement of the previous work of Chen et al. [7], named Diogenes [8], is the most recent result in the area. From a scalability point of view, Diogenes [8] is the first MPC implementation for a non-trivial task that scales to thousands of parties. To achieve efficiency and scalability, Diogenes [8] deploys the so-called *coordinator model*, which is a setting consisting of a powerful coordinator, and thousands of relatively lightweight computation parties. The RSA modulus generation protocol proposed in Diogenes [8] is secure against $n-1$ (out of the n) malicious parties, and the coordinator. Although security of the protocol is proven against a semi-honest coordinator, Diogenes [8] is actually secure with identifiable abort for any active corruption of the coordinator, and upto $n-1$ parties, as the coordinator's actions can be verified by a public bulletin board construction. Building upon the work of Chen et al. [7], Diogenes [8] also deploys the CRT representation, and constructive sampling techniques. To avoid the communication cost that pairwise messages incur, and to exploit the potential of packing and SIMD to the fullest, Diogenes [8] is based on a Ring LWE addi-

tively homomorphic encryption scheme (AHE), where the coordinator is tasked to perform all the homomorphic additions necessary, as well as relay messages. Malicious security for such a large scale application, is achieved by a composition of zero-knowledge techniques, the certification of which is aggregated and verified only once, at the end of the protocol, and only for the successful protocol iteration. In Table 1 the main functionality and security features of our work and the related works are summarized.

Protocol	**Security**	**Dishonest Majority**	**#Parties**	**Test**	**No Leakage**
[5]	Passive	×	$n \geq 3$	biprimality	✓
[16]	Active	×	$n \geq 3$	biprimality	✓
[31]	Active	✓	$n = 2$	biprimality	×
[20]	Passive	✓	$n = 2$	biprimality	✓
[1]	Passive	×	$n \geq 3$	primality	✓
[12]	Active	×	$n = 3^*$	primality	✓
[22] [23]	Active	✓	$n \geq 2$	biprimality	✓
[17]	Active	✓	$n = 2$	biprimality	×
[7]	Active	✓	$n \geq 2$	biprimality	✓
[8]	Active**	✓	$n \geq 2$	biprimality	✓
Ours	Active	✓	$n \geq 2$	biprimality	✓

Table 1. Comparison of the related work.

1.2 Our Contribution

In this work we show how to securely generate an RSA biprime in the standard multiparty setting, where all parties contribute equally to the computation. We assume a static active adversary who can corrupt up to $n - 1$ (out of the total n) parties, but remark that our proposal works with generic MPC, allowing the deployment of different security models, based on the needs of the application at hand. This makes our protocol MPC-platform-independent, as it can be realized with any MPC technology that is based on linear secret sharing techniques. For example, Shamir's secret sharing [37] can be deployed, if our goal is to produce the RSA moduli in the honest majority setting; or (a variant of) the replicated secret sharing scheme of Araki et al. [2,18], should high throughput be the main goal of the MPC implementation.

*The protocol can be non-trivially extended to support more than 3 parties, but efficiency does not scale.

**Diogenes achieves the stronger notion of (active) security with identifiable abort.

Following the paradigm of recent work [7], we design a constructive distributed sampling sub-protocol that increases the probability of our overall protocol generating a biprime. Crucially, we achieve this constructive sampling having the parties first sample multiplicative sharings of a certain form, and then transforming them into additive sharings, by computing their product in a semi-honest fashion. This does not degrade the security of the RSA generation protocol, because subsequently we reveal the public biprime N (i.e., the product of the sampled candidate primes p and q). An adversary who succeeds in introducing an additive error in the sharings of p or q that is consistent with the error in their product N, should effectively factor N, which is hard by the original assumption for an RSA biprime. This semi-honest multiplication presents itself as a major bulk of the protocol's cost, so the savings from performing it semi-honestly are substantial.

Another important technique we deploy is to run the biprimality test in terms of checking the Jacobi symbol (which identifies most of the biprimes successfully) without checking the consistency of the input shares in the test. Note that the Jacobi test has to be repeated once for each candidate and sec times for a candidate that passes the first iteration of the test, for sec a statistical security parameter. This means that any cost savings in this part of the protocol impact significantly the overall efficiency. The more computationally and communication intensive consistency checks are only performed on the candidate for which all repetitions of the Jacobi test have succeeded.

To perform the consistency check that follows the Jacobi symbol test, our protocol requires to convert a bounded additive sharing from its CRT representation to a single additive sharing over the integers. This is to match the computations performed in the exponent over the integers for the completion of the Jacobi test. We design a protocol that performs the aforementioned conversion, and we remark that in addition to being necessary for our RSA modulus generation, this protocol is of independent interest. For example, one of the PRF constructions in Grassi et al. [21], requires the MPC preprocessing field to be compatible with an elliptic curve group $\mathbb{G}$. Our exponentiation protocol, with public output and secret exponent, would make their preprocessing field compatible with the latest SHE techniques [3, 28], since those require special primes, which might be incompatible with elliptic curve groups. Our work might also improve the preprocessing efficiency in other works, which need to compute g^x in public, where x is secret shared [10, 24, 38], but we leave this for future investigation.

Lastly, we analyze our protocol, and set concrete parameters to compute the communication cost it incurs. We also show how the communication cost of our protocol scales in the number of parties, and for different parameter sets. With conservative estimations, and a statistical security parameter set to $\sigma = 80$, our

protocol outperforms the current state-of-the-art [7] in all but one settings: the semi-honest security with 16 parties setting. For malicious security, and primes of 2048 bits, our protocol improves the previous work by over 30 times, both in the two-party, and in the 16-party case.

To summarize, our main contributions are as follows:

1. Our RSA modulus generation protocol works for generic MPC, being able to leverage any MPC technology based on linear secret sharing.

2. We constructively sample candidate primes, transforming multiplicative sharings to additive sharings, by computing their product in a semi-honest fashion, which is checked for maliciously inserted additive errors later in the protocol, resulting in the protocol's cost reduction.

3. The first biprimality check, implemented by means of checking the Jacobi symbol, is costly and repeated sec times in our protocol. We show how to postpone the even costlier consistency check on the shares contributed to the Jacobi test, in order to again gain efficiency.

4. We design a protocol to convert an additive sharing over a ring to an additive sharing over the integers, which is of independent interest.

5. We demonstrate that our protocol improves the communication cost over the current state-of-the-art [7].

1.3 Technical Overview

Our main protocol, Π_{RSAGen}, works in five distinct phases: (1) the sampling phase, aiming at generating two prime numbers p and q, secret shared among the protocol participants; (2) the combine phase, computing the product N of the previously sampled candidate primes, which is securely computed and then revealed to all parties; (3) the Jacobi test, checking whether the computed N is a biprime; (4) the consistency check, ensuring input consistency in the presence of malicious adversaries, should the Jacobi test indicate a candidate biprime; and (5) the GCD test, which checks again whether N is a biprime, to ensure that the protocol did not accept a false positive that the Jacobi test may not catch.

Our Sampling phase first deploys a technique similar to the one introduced by Malkin et al. [29], which they term *distributed sieving*, and variations thereof have been also deployed in more recent works [7, 8, 22, 23]. Distributed sieving entails each party sampling a multiplicative share for each of the two primes p and q, then performing a (semi-honest) multiplication on these shares, and then re-share them to transform them into additive shares. With the distributed

sieving we increase the probability of sampling primes p and q. Similarly to recent related work [7, 8], we leverage the Chinese Remainder Theorem (CRT) to further increase the efficiency of our protocol. To this end, we show how to extend the standard actively secure MPC functionality to work on separate MPC engines: one for each of the CRT components we consider. We call this functionality $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$.

In the Combine phase of Π_{RSAGen}, based on the aforementioned $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$ functionality, we perform an actively secure multiplication between the two sampled primes, we reveal the result to all parties, and check whether the product falls within the predetermined bounds, and whether it is coprime to a value M_{sample}, which is the product of ℓ_1 primes. Should both of these checks pass, the combine phase is completed and we proceed to the Jacobi test on the generated candidate biprime.

The Jacobi test aims at establishing whether the product N is a biprime. Although this test introduces no false negatives, it has a probability of $1/2$ of introducing a false positive (i.e., accepting a non-biprime). To increase the probability of N being a biprime to $2^{-\mathsf{sec}}$ (before proceeding to the ultimate GCD test) we repeat the Jacobi test sec times. The core of the Jacobi test lies in a secure exponentiation protocol, with public output, where the computations in the exponent are performed over the integers. We deploy the exponentiation protocol proposed by Grassi et al. [21] to compute the desired Jacobi symbol. If the Jacobi symbol is ± 1, we proceed to the next phase, which is the Consistency Check.

The consistency check ensures that the protocol will abort, in the presence of active adversaries who have input inconsistent shares of the candidate primes. To achieve this, we carefully mask the exponent of the Jacobi test, with bounded randomness (for which we have devised a specialized protocol, Π_{Rand2k}) so that all computations are performed over the integers without wrap around. Then, the masked value itself needs to be an additive sharing over the integers. To this end, we have devised a protocol to convert an additive sharing over a ring, into an additive sharing over the integers, named Π_{ConvInt}. By ensuring that indeed no computation wrapped around, we check an equivalent relationship for the exponentiation performed for the Jacobi symbol computation, which serves as a proof of input consistency of the shares contributed by each party to the Jacobi test. This ensures security against malicious adversaries.

The last phase of our protocol aims at eliminating any false positives that are not filtered out by the Jacobi test. Concretely, in the GCD phase we wish to verify that $\gcd(N, [\![p + q + 1]\!]) = 1$. This phase requires again the generation of bounded randomness, for which we deploy the same protocol we devised for the Jacobi test, as well as a careful selection of the bounds, and number of

necessary CRT components, ensuring that no wrap around happens during the secure computations. Note that according to the original work of Boneh and Franklin [5], the latter test introduces a false negative, in the particular case of $N = p \cdot q$, with p, q primes, and $q = 1 \bmod p$.

2 Preliminaries

2.1 Chinese Remainder Theorem - CRT

Following the blueprint of the two most recent works in distributed RSA modulus generation [7, 8], we deploy in our work the Chinese Remainder Theorem to increase the efficiency of our protocol. We recall here the Chinese Remainder Theorem [25].

Theorem 1. *Let $N = pq$ where p and q are relatively prime. Then $\mathbb{Z}_N \simeq \mathbb{Z}_p \times \mathbb{Z}_q$ and $\mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$. Moreover, let f be the function mapping elements $x \in \{0, \ldots, N-1\}$ to pairs (x_p, x_q) with $x_p \in \{0, \ldots, p-1\}$ and $x_q \in \{0, \ldots, q-1\}$ defined by $f(x) = ([x \bmod p], [x \bmod q])$. Then f is an isomorphism from $\mathbb{Z}_N$ to $\mathbb{Z}_p \times \mathbb{Z}_q$, as well as an isomorphism from $\mathbb{Z}_N^*$ to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$.*

The CRT generalizes to any vector of pairwise relatively primes $p_1, p_2, \ldots, p_\ell$, whose product is $N = \prod_{i=1}^{\ell} p_i$. Then the function f mapping elements $x \in \{0, \ldots, N-1\}$ to tuples $(x_{p_1}, \ldots, x_{p_\ell})$ with $x_{p_j} \in \{0, \ldots, p_j - 1\}$, is an isomorphism from $\mathbb{Z}_N$ to $\mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_\ell}$ and from $\mathbb{Z}_N^*$ to $\mathbb{Z}_{p_1}^* \times \cdots \times \mathbb{Z}_{p_\ell}^*$. We refer to the tuples $(x_{p_1}, \ldots, x_{p_\ell})$ as the CRT representation of x.

To convert an element from its CRT representation to its representation mod N, we deploy the so-called CRT Reconstruction algorithm, which is presented in Fig. 1.

Algorithm $\mathsf{CRTrec}((x_{p_1}, \ldots, x_{p_\ell}), (p_1, \ldots, p_\ell))$

1. Compute $N = \prod_{i=1}^{\ell} p_i$.
2. For all $i \in \{1, \ldots, \ell\}$ compute $N_i = N/p_i$ and find M_i satisfying $N_i \cdot M_i = 1 \bmod p_i$.
3. Compute $x = \sum_{i=1}^{\ell} x_{p_i} N_i M_i \bmod N$.

Fig. 1. CRT Reconstruction Algorithm.

2.2 Notation

We define $M_{\mathsf{sample}} = 3 \cdot 5 \cdot \cdots \cdot p_b$ to be the product of the first b primes (excluding 2). This is the space from which we sample the first multiplicative sharings of the

candidate primes p and q in our protocol. Further, we define $M_\ell = p_1 \cdot p_2 \cdots p_\ell$ to be the product of ℓ distinct primes of size 128 bits each. To achieve efficient arithmetic over M_ℓ we use ℓ distinct MPC engines, each of which operates over p_i. At different stages of our protocols we work either with these distinct MPC engines, or we perform the CRT reconstruction of the variables we work with over an MPC engine M_ℓ.

To compress and simplify notation throughout this paper, we denote by (x, ℓ) the CRT representation of $x \bmod M_\ell$, that is all ℓ CRT components $(x \bmod p_1, \ldots, x \bmod p_\ell)$. The local operation of CRT reconstruction of $x \bmod M_\ell$ from its CRT representation is denoted as $\mathsf{CRTrec}(x, \ell)$.

We use square brackets to denote additively secret shared values, e.g., the shared version of x is denoted by $[x]$. We use double square brackets for the *authenticated* secret shared values, e.g., the authenticated shared version of x is denoted by $[\![x]\!]$. When the sharings are over the CRT representation with ℓ CRT components, we denote the sharings as $[x, \ell]$, and $[\![x, \ell]\!]$, respectively, and assume ℓ MPC engines operating in parallel, one for each CRT component.

3 Protocol Ingredients

Our main protocol for the biprime generation depends on several functionalities. In this section we present all functionalities that are necessary for the realization of the final $\mathcal{F}_{\mathsf{RSAGen}}$ functionality, and elaborate on the non-standard ones. We begin the description of the ingredients that comprise our final protocol with a roadmap explaining the dependencies between the functionalities required to realize $\mathcal{F}_{\mathsf{RSAGen}}$.

3.1 Roadmap

In Fig. 2 we demonstrate the functionality dependencies for the RSA modulus generation. We denote functionalities with circles, and protocols with rectangles. On the dependency vectors 'H' stands for hybrid (as in which hybrid model do we assume for the protocol), and 'R' stands for realizing, and leads to the functionality that the origin protocol realizes. In this section we show how to reach the root of the depicted tree, namely the Π_{RSAGen} protocol, which in turn realizes the $\mathcal{F}_{\mathsf{RSAGen}}$ functionality.

The first functionality that our protocol makes use of, is the $\mathcal{F}_{\mathsf{ABBWithErrors}}$. This is used in the sampling phase of Π_{RSAGen}, where we resort to a semi-honest multiplication protocol to compute the additive shares of the two primes contributed by each party, from their multiplicative shares. This is realized by the $\Pi_{\mathsf{ABBWithErrors}}$ protocol, which in turn is constructed in the $\mathcal{F}_{\mathsf{ABBWithErrors-Prep}}$-hybrid model (realized by $\Pi_{\mathsf{ABBWithErrors-Prep}}$). The reader can think of this func-

tionality as the standard MPC arithmertic black-box, secure against passive adversaries. The preprocessing phase of the arithmetic black-box produces unauthenticated input tuples, and multiplication triples. We elaborate on the workings of $\Pi_{\mathsf{ABBWithErrors}}$ in Section 3.2.

Then, the rest of the sampling phase, as well as the combining phase of Π_{RSAGen} uses the $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$ functionality, realized by the $\Pi_{\mathsf{MPC\text{-}CRT}}$ protocol, which is in turn designed in the standard $\mathcal{F}_{\mathsf{MPC}}$-hybrid model. For completeness, we present the $\mathcal{F}_{\mathsf{MPC}}$ functionality in Fig. 17, Appendix B. In Section 3.3 we show how to generalize the standard actively secure MPC functionality to support parallel MPC engines operating over sharings of the CRT representation of the inputs, designing therefore the $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$ functionality.

The Jacobi test phase of Π_{RSAGen} makes use of the standard broadcast, and randomness sampling functionalities, which are presented for completeness in Appendix B, Fig. 18, and Fig. 19, respectively. The consistency check that follows the Jacobi test of Π_{RSAGen} requires two additional functionalities. To support these two additional functionalities, we augment the $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$ functionality with two additional commands, and integrate them into the $\Pi_{\mathsf{AdvMPC-CRT}}$ protocol, realizing the corresponding $\mathcal{F}_{\mathsf{AdvMPC-CRT}}$ functionality. This is presented in Section 3.4. Concretely, the first command implements a functionality that generates bounded randomness to accommodate computations that would otherwise wrap around in the original CRT representation. This construction is presented in Section 3.4 and the protocol that realizes uses the additional $\mathcal{F}_{\mathsf{maBits}}$ command of the $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$ functionality. The latter functionality facilitates the generation of multiply authenticated random bits [35]. Furthermore the consistency check that follows the Jacobi test requires certain computations to be performed over the integers. To realize this second command we need to convert a sharing from its CRT representation to an additive sharing of the CRT reconstructed value over the integers. We explain how to achieve this in Section 3.4.

3.2 Unauthenticated Arithmetic Black Box Functionality

$\mathcal{F}_{\mathsf{ABBWithErrors}}$ (Fig. 3) is the functionality implementing an unauthenticated arithmetic black box MPC. Our Π_{RSAGen} protocol makes use of this functionality to perform a multiplication, in a more efficient manner than the actively-secure version. This does not cause the overall security of our protocol to depreciate, because the range in which the parties' inputs lie are implicitly checked when opening the product of the two sampled candidate primes, and the remaining primitives used in Π_{RSAGen} are actively secure.

For completeness, we detail the protocol realizing the unauthenticated ABB functionality, $\Pi_{\mathsf{ABBWithErrors}}$, in Appendix A, Fig. 14. The $\Pi_{\mathsf{ABBWithErrors}}$ protocol implements the online phase of the unauthenticated arithmetic black box, and

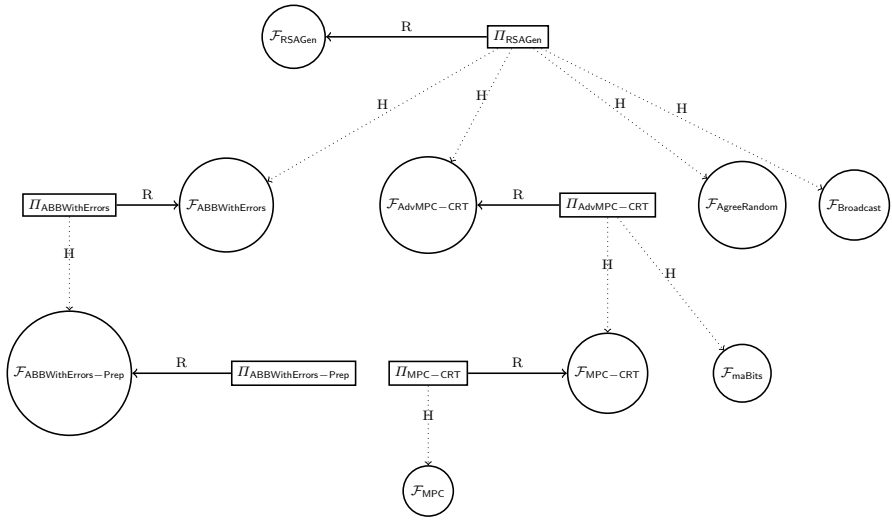


Fig. 2. Functionality dependencies for RSA modulus generation.

it works in the $\mathcal{F}_{\mathsf{ABBWithErrors-Prep}}$-hybrid model. This functionality, realized by $\Pi_{\mathsf{ABBWithErrors-Prep}}$, is used to generate the necessary preprocessing material for the online phase. Concretely, the required preprocessing material is (unauthenticated) input tuples and multiplication triples. The protocol for the preprocessing for tuples is listed in $\Pi_{\mathsf{InputTuple}}$ (Fig. 15), while the protocol for the preprocessing of triples is listed in $\Pi_{\mathsf{TripleGeneration}}$ (Fig. 16), in Appendix A. Note that the $\Pi_{\mathsf{TripleGeneration}}$ protocol makes use of the standard $\mathcal{F}_{\mathsf{Rand}}$ functionality, which is presented for completeness in Fig. 20, Appendix B.

For simplicity and clarity of presentation we present here (and in Appendices A and B) the protocols implementing the standard unauthenticated arithmetic black box functionality. We also invoke the corresponding functionality in our protocol in the usual manner. However, we recommend this functionality to be implemented over a CRT representation of the sharings and inputs, meaning that we would need multiple MPC engines operating in parallel for each CRT component. We detail how to achieve the standard actively secure MPC functionality over CRT components in Section 3.3. The $\mathcal{F}_{\mathsf{ABBWithErrors}}$ functionality can be also implemented over CRT components in the same manner. We have assumed the aforemenioned implementation of the $\mathcal{F}_{\mathsf{ABBWithErrors}}$ functionality for the efficiency analysis of our protocol.

Functionality $\mathcal{F}_{\mathsf{ABBWithErrors}}$

Initialize: Parties call $\mathcal{F}_{\mathsf{ABBWithErrors-Prep}}$ to receive preprocessing tuples and triples.
Input: Receive a value x from some party and store x.
Mult$([x],[y])$**:** Await for Δ from the adversary. Compute $z = (x \cdot y) + \Delta$ and store $[z]$.
Share$([x])$**:** For each corrupt party $i \in \mathcal{A}$ receive x_i from the adversary. Sample uniformly honest parties' shares $x_{j\, j \notin \mathcal{A}}$ such that $\sum_{i=1}^{n} x_i = x$. Send x_i to P_i.

Fig. 3. Arithmetic Black Box Functionality with Errors.

3.3 MPC on CRT Components

In this subsection we describe a new functionality and its associated protocol to perform secure multiparty computation over a big composite modulus, by relying on the Chinese Remainder Theorem. The functionality $\mathcal{F}_{\mathsf{MPC-CRT}}$ (Fig. 4) essentially implements the standard MPC functionality, but on sharings in their CRT representation. To accommodate computations on this type of sharings, we deploy ℓ MPC engines, for ℓ the maximum possible number of CRT components in the representation, as shown in $\Pi_{\mathsf{MPC-CRT}}$ (Fig. 5). Each of these MPC engines operates over one of the prime moduli of the CRT representation, and each of these ℓ prime moduli is 128-bits long. All ℓ MPC engines operate in parallel, in a much smaller space than the big composite modulus over which the final reconstruction is performed. This has a profound impact on the efficiency of our Π_{RSAGen} protocol.

Functionality $\mathcal{F}_{\mathsf{MPC-CRT}}$

Let $[x, \ell]$ denote the identifiers for the ℓ components of the CRT representation a value x stored in the functionality. Let $A \subset \{1, \ldots, n\}$ denote the index set of the corrupted parties.
Init: Receive $p_1, \ldots, p_\ell$ primes from all parties, store them and compute $M_\ell = \Pi_{i=1}^{\ell} p_i$.
Input: Receive a tuple $(x, \ell') \in \mathbb{Z}_{M_{\ell'}}$ with $\ell' \leq \ell$ from some party and store $([x, \ell'])$.
Add$([x,\ell'],[y,\ell'])$**:** Check if (x, ℓ') and (y, ℓ') exist in memory. If so compute $z = x + y \bmod M_{\ell'}$ and store $([z, \ell'])$.
Mult$([x,\ell'],[y,\ell'])$**:** Check if (x, ℓ') and (y, ℓ') exist in memory. If so, compute $z = x \cdot y \bmod M_{\ell'}$ and store $([z, \ell'])$.
Open$([x,\ell'])$**:** Check if (x, ℓ') exists in memory.If so send the value x to all parties.
OpenTo$([x,\ell'],j)$**:** Check if (x, ℓ') exists in memory. If so send the CRT represented values (x, ℓ') to party P_j.

Fig. 4. MPC over CRT Functionality

$\Pi_{\mathsf{MPC-CRT}}$

Init(ℓ)**:** To initialize ℓ CRT MPC engines, parties call $\mathcal{F}_{\mathsf{MPC}}.\mathsf{Init}(\mathbb{F}_{m_i})\ \forall\ \ell$ primes $[m_1, m_2, \ldots, m_\ell]$.
Input(x, ℓ')**:** To provide an input $x \in \mathbb{Z}_{M_{\ell'}}, \ell' \leq \ell$, any party calls $\mathcal{F}_{\mathsf{MPC}}.\mathsf{Input}(x \bmod m_i)\ \forall\ \ell'$ primes $[m_1, m_2, \ldots, m_{\ell'}]$, and defines $[\![x, \ell']\!]$ as $((x, \ell')^{(1)}, (x, \ell')^{(2)}, \ldots, (x, \ell')^{(n)})$, where each $(x, \ell')^{(j)}$ represents the ℓ' CRT shares that each player j obtains.
Add($[\![z, \ell']\!], [\![x, \ell']\!], [\![y, \ell']\!]$)**:** To add two shared values $[\![x, \ell']\!], [\![y, \ell']\!]$ parties call $\mathcal{F}_{\mathsf{MPC}}.\mathsf{Add}([\![z, \ell_i]\!], [\![x, \ell_i]\!], [\![y, \ell_i]\!])\ \forall\ \ell'$ MPC engines operating over $[m_1, m_2, \ldots, m_{\ell'}]$, and set $[\![z, \ell']\!] \leftarrow [\![x, \ell']\!] + [\![y, \ell']\!]$.
Mult($[\![z, \ell']\!], [\![x, \ell']\!], [\![y, \ell']\!]$)**:** To multiply two shared values $[\![x, \ell']\!], [\![y, \ell']\!]$ parties call $\mathcal{F}_{\mathsf{MPC}}.\mathsf{Mult}([\![z, \ell_i]\!], [\![x, \ell_i]\!], [\![y, \ell_i]\!])\ \forall\ \ell'$ MPC engines operating over $[m_1, m_2, \ldots, m_{\ell'}]$, and set $[\![z, \ell']\!] \leftarrow [\![x, \ell']\!] \cdot [\![y, \ell']\!]$.
Open($[\![x, \ell']\!]$)**:** To open a shared value $[\![x, \ell']\!]$ all parties call $\mathcal{F}_{\mathsf{MPC}}.\mathsf{Open}([\![x, \ell_i]\!])\ \forall\ \ell'$ MPC engines operating over $[m_1, m_2, \ldots, m_{\ell'}]$, and receive (x, ℓ').
OpenTo($[\![x, \ell']\!], j$)**:** To open a shared value $[\![x, \ell']\!]$ party P_j calls $\mathcal{F}_{\mathsf{MPC}}.\mathsf{OpenTo}([\![x, \ell_i]\!], j)\ \forall\ \ell'$ MPC engines operating over $[m_1, m_2, \ldots, m_{\ell'}]$, and receives (x, ℓ').

Fig. 5. Protocol for arithmetic MPC over CRTmoduli.

3.4 Advanced MPC CRT

Functionality $\mathcal{F}_{\mathsf{AdvMPC-CRT}}$

This functionality reproduces all the commands of $\mathcal{F}_{\mathsf{MPC-CRT}}$ and extends it with:
Rand2k(ℓ', k)**:** Sample $r \xleftarrow{\$} \mathbb{Z}_{2^k}$ and store $([r, \ell'])$, for $\ell' \leq \ell$.
ConvInt($[x, \ell']$)**:** Check if (x, ℓ') exists in memory. If so, uniformly sample shares $x^{(i)}_{\mathsf{Int}} \in \mathbb{Z}$ for each party P_i s.t. $\sum_{i=1}^{n} x^{(i)}_{\mathsf{Int}} = x$, and send them to the corresponding parties. (Note: the sum is taken in $\mathbb{Z}$).
LevelUp($[p, \ell], [q, \ell], \ell'$)**:** Receive $p_{\ell+1}, \ldots, p_{\ell'}$ from all parties, store them and compute $M_{\ell'} = \prod_{i=1}^{\ell'} p_i$. Store $([p, \ell'])$ and $([q, \ell'])$.

Fig. 6. Advanced MPC over CRT Functionality

The functionality $\mathcal{F}_{\mathsf{MPC-CRT}}$ (Fig. 4) is similar to the *classic* MPC functionality, but over a direct product of finite fields. We also define $\mathcal{F}_{\mathsf{AdvMPC-CRT}}$, which is the functionality $\mathcal{F}_{\mathsf{MPC-CRT}}$ augmented with three additional commands. The first one is the Rand2k command, which samples a random secret shared value $r < 2^k$ in its CRT representation over ℓ' moduli, used in our Π_{RSAGen} protocol to ensure no overflows during computation. The second one is the ConvInt command, which allows the parties to convert a CRT sharing to an integer sharing

of the same value. The third one is the LevelUp command, which extends the CRT representation of the sharings of the candidate primes p and q to the CRT representation of the same sharings, but with additional CRT components. This augmented functionality $\mathcal{F}_{\mathsf{AdvMPC-CRT}}$ is described in Fig. 6. We subsequently detail how the three additional commands are realized.

Bounded Randomness in Shared CRT Representation. The Rand2k command allows us to sample a random CRT sharing, the reconstruction of which falls within a predetermined range. This is necessary in our main protocol to accommodate computations that would otherwise overflow over the intial (smaller) CRT representation. The protocol implementing the Rand2k command is listed in Π_{Rand2k} (Fig. 8), and it uses an additional command of the $\mathcal{F}_{\mathsf{MPC-CRT}}$ functionality, namely the $\mathcal{F}_{\mathsf{maBits}}$ command. The $\mathcal{F}_{\mathsf{maBits}}$ command itself, presented in Fig. 7, is a slightly different version of the one presented by Rotaru et al. [35]. In our case, we modify the command so that it outputs the integer sharing of the bit, which was discarded in the original paper. We need this integer sharing later, in the Π_{ConvInt} protocol.

Functionality $\mathcal{F}_{\mathsf{maBits}}$

1. For $i = 1, \ldots, m$ the functionality calls $\mathcal{F}_{\mathsf{MPC}}.\mathsf{GenBit}()$ so as to store a bit b_i.
2. The bits b_i are retrieved from $\mathcal{F}_{\mathsf{MPC}}$ and are enterred into the $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$ functionality.
3. The functionality samples a sharing of b_i in $\mathbb{Z}_p$ and send its share $b_i^{(j)}$ to every party P_j. It also publicly outputs $k_i = \lfloor \frac{\sum b_i^{(j)}}{p} \rfloor$.
4. The functionality waits for a message Abort or Ok from the adversary. If the message is Ok then it continues.

Fig. 7. The ideal command for generating random bits

Π_{Rand2k}

Rand2k: On input (ℓ', k), to generate a random CRT sharing $[\![r, \ell']\!]$ with $r < 2^k$, parties do the following:

1. All parties call $\mathcal{F}_{\mathsf{maBits}}$ to generate k random bits $\{[\![b_i, \ell']\!]\}_{i\in[k]}$ shared across all MPC engines, and receive k_i which gives an integer sharing of b_i w.r.t. the first CRT component of $[\![b_i, \ell']\!]$.
2. All parties compute $[\![r, \ell']\!] = \sum_{i\in[k]} 2^i \cdot [\![b_i, \ell']\!]$.
3. Output $[\![r, \ell']\!]$ and the integer sharing $[r_{\mathsf{Int}}]$.

Fig. 8. Protocol for generating a random CRT sharing $[\![r, \ell']\!]$, which CRT reconstructs to a bounded random value $r < 2^k$, and the corresponding integer sharing

Converting a CRT Sharing to an Integer Sharing. Our main protocol requires a command which converts a CRT sharing to an integer sharing of the same underlying secret. This is necessary during the Jacobi test of Π_{RSAGen}, because we need to ensure that all computations in the exponent are performed over the integers, and hence the shares in the exponent are also reconstructed over the integers; otherwise, the correctness of the protocol is not guaranteed due to potential wrap around. To ensure that indeed the computations are done over the integers, we realize the $\mathsf{ConvInt}$ command with the protocol Π_{ConvInt}, listed in Fig. 9. This protocol allows the parties to obtain an unauthenticated integer sharing of the CRT sharings they already hold, without leaking any information about the underlying secret value.

The execution of the protocol Π_{ConvInt} does not leak any information about the secret. Indeed, the only opened value in the protocol is $[\![x, \ell_{\mathsf{Jac}}]\!] + [\![r, \ell_{\mathsf{Jac}}]\!]$ with $r \xleftarrow{\$} [2^{B+\mathsf{sec}}]$ and $x < 2^B$. Therefore, by Theorem 2, we have that the distance between the distribution of this opened value and the uniform distribution is upper bounded by $2^{-\mathsf{sec}}$.

This protocol only produces an *unauthenticated* integer sharing, but the consistency of the shares is checked later in the Π_{RSAGen} protocol. During the broadcast at step 5 of the Consistency Check, a malicious adversary can broadcast any arbitrary value, but if the value of the shared secret would be altered by the adversary's broadcast, then the equality check which follows will fail with probability 1/2.

Π_{ConvInt}

Let $B \in \mathbb{N}$ be an upper bound for the bit-length of the input; i.e. for any input x, we assume $x < 2^B$. We select ℓ_{Jac}, such that $M_{\ell_{\mathsf{Jac}}} = \prod_{i=1}^{\ell_{\mathsf{Jac}}} m_i$ is the minimal moduli product bigger than $2^{B+\mathsf{sec}+1}$

ConvInt: On input $[\![x, \ell_{\mathsf{Jac}}]\!]$, to convert the CRT sharing $[\![x, \ell_{\mathsf{Jac}}]\!]$ to an integer sharing $[x_{\mathsf{Int}}]$ that reconstructs to the same underlying secret, parties do the following:

1. Parties call $\mathsf{Rand2k}$ with input $(\ell_{\mathsf{Jac}}, B + \mathsf{sec})$ to get a CRT sharing $[\![r, \ell_{\mathsf{Jac}}]\!]$ and an integer sharing $[r_{\mathsf{Int}}]$ of a random value r, with $r < 2^{B+\mathsf{sec}}$.
2. Parties call $(t, \ell_{\mathsf{Jac}}) = \mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Open}([\![x, \ell_{\mathsf{Jac}}]\!] + [\![r, \ell_{\mathsf{Jac}}]\!])$ and do the local CRT reconstruction $t = \mathsf{CRTrec}(t, \ell_{\mathsf{Jac}})$.
3. To obtain an integer sharing of x parties locally compute $x^{(i)}_{\mathsf{Int}} = t - r^{(i)}$.
4. Parties store $x^{(i)}_{\mathsf{Int}}$ as their integer share of x.

Fig. 9. Protocol for converting a bounded CRT sharing to an integer sharing.

Theorem 2 ([36, Appendix A]). *Let M and K be positive integers, where $M \leq K$. Let the random variable X take values from $\{0, \ldots, M-1\}$ and let the random variables U be uniform on $\{0, \ldots, K-1\}$. Then $\Delta(U, X+U) \leq (M-1)/K$ is an upper bound for the distance between the two distributions.*

Extending the CRT Representation. The $\mathsf{LevelUp}$ command extends the CRT representation of $[p, \ell]$ and $[q, \ell]$, allowing us to compute over $M_{\ell'} > M_\ell$ for these two values. In our Π_{RSAGen} protocol, we use $\mathsf{LevelUp}$ command whenever a new operation on p and q could overflow the current CRT modulus. This happens twice: first during the consistency check, and then in the GCD test. We note that we execute this command only on $[p, \ell]$ and $[q, \ell]$, which have passed the biprimality test, and for which the product N is publicly known. We use both of these properties in the Π_{LevelUp} protocol (Fig. 10), which implements the $\mathsf{LevelUp}$ command.

Π_{LevelUp}

For the execution of this protocol, we assume that the parties have access to an integer sharing of the values they wish to extend, and that the product of the values to be reshared has passed the biprimality test.

LevelUp: On input $([\![p, \ell]\!], [\![q, \ell]\!], \ell')$, with $\ell' > \ell$, publicly known $N = p \cdot q$ parties do the following:

1. Each party P_j retrieves its integer sharings $p^{(j)}$ and $q^{(j)}$ of $[\![p, \ell]\!]$ and $[\![q, \ell]\!]$.
2. Each party P_j calls $\mathcal{F}_{\mathsf{MPC}}.\mathsf{Input}(p^{(j)} \bmod m_i)$ and $\mathcal{F}_{\mathsf{MPC}}.\mathsf{Input}(q^{(j)} \bmod m_i)$ for $i \in \{\ell + 1, \ldots, \ell'\}$.
3. All parties call $\mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Add}$ to obtain $[\![p, \ell']\!] = \sum_{j=1}^{n} [\![p^{(j)}, \ell']\!]$ and $[\![q, \ell']\!] = \sum_{j=1}^{n} [\![q^{(j)}, \ell']\!]$.
4. All parties call $[\![N, \ell']\!] = \mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Mult}([\![p, \ell']\!], [\![q, \ell']\!])$ and then call $\mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Open}([\![N, \ell']\!])$, abort if the result is not equal to the publicly known N.

Fig. 10. Protocol for extending the CRT representation of $[\![p, \ell]\!]$ and $[\![q, \ell]\!]$.

The protocol ensures that the new $[\![p, \ell']\!]$ and $[\![q, \ell']\!]$ are sharings of the same value as $[\![p, \ell]\!]$ and $[\![q, \ell]\!]$. Indeed, checking that both multiply to N acts as a MAC check. To be successful, a cheating adversary would have to introduce additive errors on $[\![p, \ell']\!] = [\![p, \ell]\!] + \Delta_p$ and $[\![q, \ell']\!] = [\![q, \ell]\!] + \Delta_q$, such that $\Delta_p \cdot q + \Delta_q \cdot p + \Delta_q \cdot \Delta_p = 0$; that is an adversary would have to guess either p or q from N. Because N is an RSA modulus, we assumed that factoring N is a hard problem, thus making Π_{LevelUp} secure.

4 Distributed Generation of an RSA Biprime

Our ideal functionality $\mathcal{F}_{\mathsf{RSAGen}}$, listed in Fig. 11, consists of 5 steps: Sample, Combine, Jacobi, Consistency Check, and GCD Test. The first step, samples two candidate primes p and q of approximately 1024 bits each which are both coprime with M_{sample}, which significantly increases our chances of selecting a prime. The second step computes the product of the previously sampled candidate primes, and checks that it lies in the expected range (respecting the aforementioned bit-length), and that it is not coprime with M_{sample}. The third step follows the blueprint of Boneh and Franklin [5], checking biprimality in a way similar to Miller-Rabin primality testing, and returing Abort, if the product computed is not a biprime. The fourth step serves as a constistency check, confirming that all parties have input consistent shares in the so-called Jacobi step above. The

last step is the GCD test, which catches some false positives that can potentially be introduced by the Jacobi test.

Functionality $\mathcal{F}_{\mathsf{RSAGen}}$

For λ the bit-length of each of the candidate primes p and q we aim to sample, and σ a statistical security parameter, let ℓ_1 be the number of primes, the product of which (M_{sample}) serves as the space over which we sample the candidate primes p and q, such that $\prod_{i=1}^{\ell_1} m_i = M_{\mathsf{sample}} > 2^{\lambda+\sigma}$. In addition, let ℓ_{c} be the number of primes, the product of which (M_{ℓ_c}) serves as the space over which we compute without overflow the product of the candidate primes p and q, such that $\prod_{i=1}^{\ell_{\mathsf{c}}} m_i = M_{\ell_{\mathsf{c}}} > 2^{2\cdot(\lambda+\sigma)}$.

1. On receiving Sample from all parties, First query $\mathcal{S}$ for the values $m_p = p \mod M_{\mathsf{sample}}$ and $m_q = q \mod M_{\mathsf{sample}}$ together with the shares $p^{(j)}, q^{(j)}$ of corrupt parties $P_j \in \mathcal{C}$. Then, uniformly sample $p^{(j)}, q^{(j)}$ and send them to honest parties $P_j \in \mathcal{H}$ with the condition that $\sum_j p^{(j)} \mod M_{\mathsf{sample}} = m_p$ and $\sum_j q^{(j)} \mod M_{\mathsf{sample}} = m_q$.
2. On receiving Combine from all parties, send $N_i = (p \cdot q) \bmod m_i$ for all $i \in [\ell_{\mathsf{c}}]$ to all parties. If $\mathsf{gcd}(N, M_{\mathsf{sample}}) \neq 1$ send $\mathsf{Abort}_{\mathsf{GCD}}$ to all parties; if $N \notin [2^{2\lambda}, 2^{2\cdot(\lambda+\sigma)}]$ send $\mathsf{Abort}_{\mathsf{OutOfRange}}$ to all parties.
3. On receiving Jacobi(γ) from all parties, First, compute $y = \gamma^{(N-p-q+1)/4} \mod N$ and send y to $\mathcal{S}$. Then, receive y' from $\mathcal{S}$ and send y' to all parties. Finally, if $y' \neq \pm 1$ then send Abort to all parties.
4. On receiving Consistency Check from all parties, if y' received during the Jacobi command was not equal to y, then send Abort to all parties.
5. On receiving GCD Test from all parties, compute $b = \mathsf{gcd}(N, (p + q - 1))$. If $b = 1$, send $(b, \mathsf{Biprime})$ to all parties, otherwise send $(b, \mathsf{Non\ Biprime})$.

Fig. 11. RSA Modulus Generation Functionality

We now concretely detail the protocol Π_{RSAGen} (Fig. 12), realizing the $\mathcal{F}_{\mathsf{RSAGen}}$ functionality. Each party P_j samples a multiplicative share $\hat{p}^{(j)}$, such that $\mathsf{gcd}(M_{\mathsf{sample}}, \hat{p}^{(j)}) = 1$. The goal of the Sampling phase is to convert the multiplicative sharing $\hat{p} = \hat{p}^{(1)} \cdots \hat{p}^{(n)}$ over the integers, into an additive sharing $p' = p'^{(1)} + \cdots + p'^{(n)}$ over $\mathbb{Z}/(M_{\mathsf{sample}}\mathbb{Z})$, such that $p' = \hat{p} \bmod M_{\mathsf{sample}} = p'^{(1)} + \cdots + p'^{(n)} = \hat{p}^{(1)} \cdots \hat{p}^{(n)} \bmod M_{\mathsf{sample}}$. So, each party P_j engages in a (semi-honest) multiplication with their secret share $\hat{p}^{(j)}$. In the end, all parties hold an additive sharing of the product $\hat{p}^{(j)} = \hat{p}_1^{(j)} + \cdots + \hat{p}_n^{(j)} \bmod M_{\mathsf{sample}}$. After the multiplication is done over $\mathbb{Z}/(M_{\mathsf{sample}}\mathbb{Z})$, parties set their local share as $p^{(j)} = p'^{(j)} + r^{(j)} \cdot M_{\mathsf{sample}}$ and use this in the CRT Input procedure, over ℓ_c CRT components, and thus ℓ_c MPC engines.

Once two candidate primes p and q have been sampled in a secret shared fashion as described above, the Combine phase begins. First, the parties sum the contributions of each party into the additive sharing, over ℓ_cCRT components. Then, the candidate biprime is computed, using an actively secure multiplication over the CRT representation of the sharings of p and q. Lastly, the parties open the resulting candidate biprime N, and each party P_j locally performs the CRT reconstruction and obtains the biprime N in the standard form. Each party checks that the biprime respects the bounds in which it should lie, and that it is not coprime to the upper bound of the sampling range.

The parties then begin the biprimality testing with the Jacobi test, which needs to be repeated 128 times. The core of the Jacobi test we design offers passive security; to achieve active security, should the Jacobi test pass, we proceed with the Consistency Check phase. This step ensures that parties cannot go undetected, if they use inconsistent sharings in the Jacobi test. To realize this, first we need to increase our computing space to avoid potential overflows. We do that by means of the LevelUp command, which allows us to receive the same sharings in a CRT representation with additional CRT components (to accommodate the computations). Concretely, we extend from ℓ_c components of the combine step, to ℓ_{Jac} components, which suffice for the correctness of the consistency check of the Jacobi. Then, using the Rand2k command, we receive bounded shared randomness in the CRT form with ℓ_{Jac} components. Using this randomness, we multiplicatively mask (guaranteed without overflow) the exponent of the Jacobi test, where the parties' shares have been contributed. This latter product is then converted from a CRT sharing with ℓ_{Jac} components to an integer sharing by calling the ConvInt command. The integer sharing is used to exponentiate the public value γ used in the Jacobi test, and it is then broadcasted. The randomness used in the masking operation is revealed, so that the parties can perform the final exponentiation of the Jacobi value computed to the power of the randomness in the clear. From the broadcasted values, the parties can also reconstruct again the masked version of the Jacobi test exponentiation. If the two latter values do not match, then some parties have input inconsistent shares, and the protocol aborts.

The last phase of our protocol is the GCD test, aiming at detecting (and discarding) any false positive biprimes that passed the Jacobi test. The GCD test is performed between the public biprime N, and the secret $[\![p+q-1]\!]$, and if their GCD equals 1, the test passes. Let $Q_{\mathsf{gcd}} > V \cdot N$, where $V = 2^{3\lambda+4\sigma}$. The goal is to output the product $a \cdot (p+q-1) + v \cdot N \bmod Q_{\mathsf{gcd}}$, where $a \xleftarrow{\$} [N]$, and then perform the gcd computation between N and $a \cdot (p+q-1) + v \cdot N$ on public values. In our case v needs to statistically mask the product between

a, which has $2(\lambda + \sigma)$ bits length, and $p + q - 1$, which has $\lambda + \sigma$ bits length. Hence, $\log_2 v = 3(\lambda + \sigma) + \sigma$. Next, $M_{\ell_{\mathsf{gcd}}}$ is computed, so that $v \cdot N$ fits Q_{gcd}, which makes $M_{\ell_{\mathsf{gcd}}} 5\lambda + 6\sigma$ bits long.

On a step by step basis, for the GCD test we use again the LevelUp command to extend the number of CRT components in our sharings of p and q. For the masking, similarly to the Jacobi test, we sample bounded randomness in CRT form with ℓ_{gcd} components, using again the Rand2k command. Before we open and reconstruct the final value $\hat{z}$, the gcd of which needs to be checked against the public biprime N, we also perform an additive masking with a bounded random value v. This ensures that no information about the sum of p and q, involved in the multiplicatively masking, can be factored out upon opening. Upon opening and reconstuction of the masked value, the final GCD test is performed, and if the open value is not coprime to N the protocol outputs abort and restarts.

Theorem 3. *The execution of the protocol Π_{RSAGen} UC-securely realizes the functionality $\mathcal{F}_{\mathsf{RSAGen}}$, in the ($\mathcal{F}_{\mathsf{ABBWithErrors}}$, $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{F}_{\mathsf{AgreeRandom}}$, $\mathcal{F}_{\mathsf{Broadcast}}$)-hybrid model with statistical security against a static, active adversary that corrupts up to $n - 1$ parties.*

Π_{RSAGen}

Sampling phase. All the steps below are done in parallel for p and q.

1. Each party P_j samples a multiplicative share $\hat{p}^{(j)}$, such that $\mathsf{gcd}(M_{\mathsf{sample}}, \hat{p}^{(j)}) = 1$.
2. Each party P_j calls $\mathcal{F}_{\mathsf{ABBWithErrors}}.\mathsf{Input}(\hat{p}^{(j)})$.
3. The parties call $\mathcal{F}_{\mathsf{ABBWithErrors}}.\mathsf{Mult}(p'', \hat{p}^{(1)}, \ldots, \hat{p}^{(n)})$.
4. Parties call $\mathcal{F}_{\mathsf{ABBWithErrors}}.\mathsf{Share}(p')$, such that P_j receives the residues of $p''^{(j)}$ for all primes in M_{sample}.
5. Parties reconstruct $p'^{(j)} = \mathsf{CRTRec}([p'^{(j)}, 3 \cdot \delta_{j,0}], [M_{\mathsf{sample}}, 4])$ where $\delta_{j,0}$ is the Kronecker delta.
6. Each party P_j samples $r^{(j)}$, and computes $p^{(j)} = p'^{(j)} + r^{(j)} \cdot 4 \cdot M_{\mathsf{sample}}$, such that $p^{(j)} \in [2^{\lambda}, 2^{\lambda+\sigma}]$, for σ a statistical security parameter.
7. Each party P_j calls $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Input}(p^{(j)}, \ell_{\mathsf{c}})$.

Combine

1. Parties call $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Sum}(\llbracket p, \ell_{\mathsf{c}} \rrbracket, \llbracket p^{(1)}, \ell_{\mathsf{c}} \rrbracket, \ldots, \llbracket p^{(n)}, \ell_{\mathsf{c}} \rrbracket)$ and $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Sum}(\llbracket q, \ell_{\mathsf{c}} \rrbracket, \llbracket q^{(1)}, \ell_{\mathsf{c}} \rrbracket, \ldots, \llbracket q^{(n)}, \ell_{\mathsf{c}} \rrbracket)$.
2. Parties call $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Mult}(\llbracket N, \ell_{\mathsf{c}} \rrbracket, \llbracket p, \ell_{\mathsf{c}} \rrbracket, \llbracket q, \ell_{\mathsf{c}} \rrbracket)$.
3. Parties call $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Open}(\llbracket N, \ell_{\mathsf{c}} \rrbracket)$.
4. Each party locally reconstructs $N = \mathsf{CRTrec}(N, \ell_{\mathsf{c}})$, checks that $N \in [2^{2\lambda}, 2^{2(\lambda+\sigma)}]$, and $GCD(M_{\mathsf{sample}}, N) = 1$, abort if false.

Jacobi test This is executed sec times (Grassi et al. fashion but carefully so that adding shares in the exponents is done over the integers).

1. Parties call $\mathcal{F}_{\mathsf{AgreeRandom}}$ to sample a public $\gamma \in \mathbb{Z}_N$. Repeat until Jacobi symbol $(\frac{\gamma}{N}) = 1$.
2. Using their integer shares of p and q, P_1 computes $y^{(1)} = \gamma^{(N-p^{(1)}-q^{(1)}+1)/4} \bmod N$ and calls $\mathcal{F}_{\mathsf{Broadcast}}(y^{(1)})$, and each party $P_j, j \neq 1$ computes $y^{(j)} = \gamma^{(-p^{(j)}-q^{(j)})/4} \bmod N$ and calls $\mathcal{F}_{\mathsf{Broadcast}}(y^{(j)})$.
3. All parties compute $y = \prod_{j=1}^{n} y^{(j)}$.
4. If $y \neq \pm 1$ Abort.

Fig. 12. RSA modulus generation protocol based on distributed sieving

Π_{RSAGen} **(continued)**

Consistency Check

1. Parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{LevelUp}$ with input $(\ell_{\mathsf{Jac}}, [\![p, \ell_c]\!], [\![q, \ell_c]\!])$ and publicly known $N = pq$, and receive $([\![p, \ell_{\mathsf{Jac}}]\!], [\![q, \ell_{\mathsf{Jac}}]\!])$
2. Parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{Rand2k}$ with input $(\ell_{\mathsf{Jac}}, \mathsf{sec})$ to get a CRT sharing $[\![x, \ell_{\mathsf{Jac}}]\!]$ of a random value x, bounded by 2^{sec}.
3. All parties call $\mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Mult}([\![t, \ell_{\mathsf{Jac}}]\!], [\![x, \ell_{\mathsf{Jac}}]\!], [\![((N-p-q+1)/4), \ell_{\mathsf{Jac}}]\!])$, where the multiplication result is actually bounded by M_{ℓ_c} and CRT shared in $M_{\ell_{\mathsf{Jac}}}$.
4. Parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{ConvInt}([\![t, \ell_{\mathsf{Jac}}]\!])$ to obtain an additive sharing of t over the integers, denoted as $[t]_{\mathsf{Int}}$.
5. Each party calls $\mathcal{F}_{\mathsf{Broadcast}}(\gamma^{t^{(j)}_{\mathsf{Int}}})$.
6. All parties call $\mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Open}([\![x, \ell_{\mathsf{Jac}}]\!])$, and compute $x = \mathsf{CRTrec}(x, \ell_{\mathsf{Jac}})$.
7. All parties locally check that $\prod_{j=1}^{n} \gamma^{t^{(j)}_{\mathsf{Int}}} = y^x$. Abort if equality fails.

GCD test

1. Parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{LevelUp}$ with input $(\ell_{\mathsf{gcd}}, [\![p, \ell_c]\!], [\![q, \ell_c]\!])$ and publicly known $N = pq$, and receive $([\![p, \ell_{\mathsf{gcd}}]\!], [\![q, \ell_{\mathsf{gcd}}]\!])$
2. Parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{Rand2k}$ with input $(\ell_{\mathsf{gcd}}, 2\lambda+2\sigma)$ to get a CRT sharing $[\![a, \ell_{\mathsf{gcd}}]\!]$ of a random value a, bounded by $2^{2\lambda+2\sigma}$.
3. All parties call $\mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Mult}([\![z, \ell_{\mathsf{gcd}}]\!], [\![a, \ell_{\mathsf{gcd}}]\!], [\![(p+q-1), \ell_{\mathsf{gcd}}]\!])$. Note that this is fine because open $N = p \cdot q$ in MPC in the first steps of candidate generation to enforce input consistency.
4. Parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{Rand2k}$ with input $(\ell_{\mathsf{gcd}}, 3\lambda+4\sigma)$ to get a CRT sharing $[\![v, \ell_{\mathsf{gcd}}]\!]$ of a random value v, bounded by $2^{3\lambda+4\sigma}$.
5. All parties call $\mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Add}([\![\hat{z}, \ell_{\mathsf{gcd}}]\!], [\![z, \ell_{\mathsf{gcd}}]\!], [\![v \cdot N, \ell_{\mathsf{gcd}}]\!])$.
6. All parties call $\mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Open}([\![\hat{z}, \ell_{\mathsf{gcd}}]\!])$, and compute $\hat{z} = \mathsf{CRTrec}(\hat{z}, \ell_{\mathsf{gcd}})$.
7. Locally check whether $\mathsf{gcd}(N, \hat{z}) = 1$. Otherwise parties output Abort and restart the protocol.

Fig. 13. RSA modulus generation protocol based on distributed sieving (continue)

Proof Sketch. Let $\mathcal{A}$ be a static malicious adversary, who interacts with the parties running Π_{RSAGen} and can corrupt up to $n-1$ parties. We construct a simulator $\mathcal{S}$, simulating the ideal functionality $\mathcal{F}_{\mathsf{RSAGen}}$, such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with $\mathcal{A}$ and the Π_{RSAGen}, or with $\mathcal{A}$

and $\mathcal{F}_{\mathsf{RSAGen}}$. Let $\mathcal{C}$ denote the set of (up to $n-1$) corrupted parties and let $\mathcal{H}$ denote the set of honest parties. The simulator $\mathcal{S}$ proceeds as follows:

Sample: The simulator performs all the steps below in parallel for p and q.

1. For each honest $P_j \in \mathcal{H}$, $\mathcal{S}$ samples $\hat{p}^{(j)}$ such that $\gcd(M_{\mathsf{sample}}, \hat{p}^{(j)}) = 1$.

2. For each honest $P_j \in \mathcal{H}$, $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{ABBWithErrors}}.\mathsf{Input}(\hat{p}^{(j)})$. For each corrupt $P_j \in \mathcal{C}$, $\mathcal{S}$ receives $\mathcal{F}_{\mathsf{ABBWithErrors}}.\mathsf{Input}(\hat{p}^{(j)})$ from $\mathcal{A}$.

3. When all parties call $\mathcal{F}_{\mathsf{ABBWithErrors}}.\mathsf{Mult}$, $\mathcal{S}$ waits for Δ_p from $\mathcal{A}$. After receiving Δ_p, $\mathcal{S}$ computes $p' = \Delta_p + \prod_{j=1}^{n} \hat{p}^{(j)}$.

4. When all parties call $\mathcal{F}_{\mathsf{ABBWithErrors}}.\mathsf{Share}(p')$, $\mathcal{S}$ receives from $\mathcal{A}$ the shares $p'^{(j)}$ for each corrupt $P_j \in \mathcal{C}$. It then samples and stores the remaining shares $p'^{(j)}$ for honest $P_j \in \mathcal{H}$ such that $p' = \sum_{j=1}^{n} p'^{(j)}$.

5. For each honest $P_j \in \mathcal{H}$, $\mathcal{S}$ samples an appropriate $r^{(j)}$ such that $p^{(j)} = p'^{(j)} + r^{(j)} \cdot M_{\mathsf{sample}}$ lies in the range $[2^{\lambda}, 2^{\lambda+\sigma}]$.

6. When each party calls $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Input}(p^{(j)}, \ell_{\mathsf{c}})$ in Step 7, $\mathcal{S}$ receives from $\mathcal{A}$ the inputs $(p^{(j)}, \ell_{\mathsf{c}})$ for each corrupt $P_j \in \mathcal{C}$. With these, $\mathcal{S}$ can reconstruct $p^{(j)}$ for each corrupt P_j and then compute $m_p = \sum_j p^{(j)} \mod M_{\mathsf{sample}}$ using also its simulated shares. It then sends Sample to $\mathcal{F}_{\mathsf{RSAGen}}$ on behalf of the corrupt parties and, when prompted, submits m_p and the $p^{(j)}$ that it reconstructed. To continue simulating the protocol, $\mathcal{S}$ inputs its own simulated $p^{(j)}$ into $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$ on behalf of the honest parties $P_j \in \mathcal{H}$.

By computing the residue of p and q modulo M_{sample} as influenced by $\mathcal{A}$ in Step 6 of the protocol, $\mathcal{S}$ ensures that the distribution of $N \bmod M_{\mathsf{sample}}$ produced by $\mathcal{F}_{\mathsf{RSAGen}}$ is identical to the one in the protocol. At this stage of the protocol, there is no transcript for $\mathcal{S}$ to simulate as the parties have only executed calls to other functionalities. We also note that the simulated shares $p^{(j)}$ are statistically close to the random shares sampled by $\mathcal{F}_{\mathsf{RSAGen}}$, as measured by Lemma 1, and identically distributed to the honest shares in a real execution.

Combine:

1. When all parties call $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Sum}([\![p, \ell_{\mathsf{c}}]\!], [\![p^{(1)}, \ell_{\mathsf{c}}]\!], \ldots, [\![p^{(n)}, \ell_{\mathsf{c}}]\!])$ and $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Sum}([\![q, \ell_{\mathsf{c}}]\!], [\![q^{(1)}, \ell_{\mathsf{c}}]\!], \ldots, [\![q^{(n)}, \ell_{\mathsf{c}}]\!])$ in Step 1, $\mathcal{S}$ internally executes the corresponding MPC sums.

2. When all parties call $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Mult}([\![N, \ell_{\mathsf{c}}]\!], [\![p, \ell_{\mathsf{c}}]\!], [\![q, \ell_{\mathsf{c}}]\!])$ in Step 2, $\mathcal{S}$ internally executes the corresponding MPC multiplications.

3. When all parties call $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Open}([\![N, \ell_{\mathsf{c}}]\!])$ in Step 3, $\mathcal{S}$ sends $\mathsf{Combine}$ to $\mathcal{F}_{\mathsf{RSAGen}}$ on behalf of the corrupt parties. Once the honest parties also send $\mathsf{Combine}$ to the functionality, $\mathcal{S}$ receives N_i for all $i \in [\ell_{\mathsf{c}}]$. To simulate the $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Open}$ instruction, $\mathcal{S}$ then sends the N_i values it received to each corrupt party. $\mathcal{S}$ also updates its internal simulations of the $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$ instances so that they hold the correct values for N.

4. If $\mathcal{S}$ receives $\mathsf{Abort}_{\mathsf{GCD}}$ or $\mathsf{Abort}_{\mathsf{OutOfRange}}$ from $\mathcal{F}_{\mathsf{RSAGen}}$, it makes the simulated honest parties also output the corresponding Abort in the protocol.

As the shares input by $\mathcal{A}$ are passed on $\mathcal{F}_{\mathsf{RSAGen}}$ for the generation of N, and as the shares simulated by $\mathcal{S}$ are statistically close to those sampled at random by $\mathcal{F}_{\mathsf{RSAGen}}$, the distribution of N output by $\mathcal{F}_{\mathsf{RSAGen}}$ is statistically close to the one produced by $\mathcal{S}$, which is itself identically distributed to those of a real execution.

Jacobi:

1. When parties call $\mathcal{F}_{\mathsf{AgreeRandom}}$, $\mathcal{S}$ simulates the sampling of the public γ.

2. The simulator then queries $\mathcal{F}_{\mathsf{RSAGen}}.\mathsf{Jacobi}(\gamma)$ and receives y.

3. To simulate the broadcast calls, $\mathcal{S}$ samples r_j at random to compute $y^{(j)} = \gamma^{r_j}$ for the honest parties $P_j \in \mathcal{H}$ and then modifies one of these shares $y^{(i)}$ for $P_i \in \mathcal{H}$ such that $y^{(i)} = y \cdot (\prod_{j \neq i} y^{(j)})^{-1}$. Here, to compute $y^{(j)}$ for the corrupt parties $P_j \in \mathcal{C}$, the simulator uses the shares $p^{(j)}, q^{(j)}$ that $\mathcal{A}$ input to $\mathcal{F}_{\mathsf{ABBWithErrors}}$ during sampling. Then $\mathcal{S}$ uses these simulated honest $y^{(j)}$'s as the broadcast values of the honest parties.

4. When the corrupt parties call $\mathcal{F}_{\mathsf{Broadcast}}(y^{(j)})$, $\mathcal{S}$ computes the new value of $y' = \prod_j y^{(j)}$ and sends it to $\mathcal{F}_{\mathsf{RSAGen}}$.

5. If $\mathcal{S}$ receives Abort from $\mathcal{F}_{\mathsf{RSAGen}}$, it makes the simulated honest parties output Abort.

Since the simulated $p^{(j)}$ and $q^{(j)}$ values that $\mathcal{S}$ holds for $P_j \in \mathcal{H}$ are statistically close to uniform, the distribution of the broadcast $y^{(j)}$ values are statistically close to the protocol and consistent with the correct Jacobi test result first output by $\mathcal{F}_{\mathsf{RSAGen}}$. If $\mathcal{A}$ cheats by using inconsistent values during its broadcast, then $\mathcal{S}$ correctly updates the result of the Jacobi test by passing the new y' to $\mathcal{F}_{\mathsf{RSAGen}}$.

Consistency Check:

1. When all parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{Rand2k}$, $\mathcal{S}$ samples a random $x < 2^{\mathrm{sec}}$ and receives the shares $(x^{(j)}, \ell_{\mathsf{Jac}})$ for each corrupt $P_j \in \mathcal{C}$ from $\mathcal{A}$. It then

samples the remaining shares $(x^{(j)}, \ell_{\mathsf{Jac}})$ for honest $P_j \in \mathcal{H}$, such that $x = \mathsf{CRTrec}(x, \ell_1) = \sum_{j=1}^{n}(x^{(j)}, \ell_{\mathsf{Jac}})$.

2. When all parties call $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Mult}([\![t, \ell_{\mathsf{Jac}}]\!], [\![x, \ell_{\mathsf{Jac}}]\!], [\![((N-p-q+1)/4), \ell_{\mathsf{Jac}}]\!])$, $\mathcal{S}$ internally executes the MPC multiplication.

3. When all parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{ConvInt}([\![t, \ell_{\mathsf{Jac}}]\!])$, $\mathcal{S}$ samples $t_{\mathsf{Int}}^{(j)}$ for each party P_j such that $t = \sum_{j=1}^{n} t_{\mathsf{Int}}^{(j)}$, and sends them.

4. To simulate the broadcast calls, $\mathcal{S}$ modifies one of the honest shares $\gamma^{t_{\mathsf{Int}}^{(i)}}$ for $P_i \in \mathcal{H}$ such that $\gamma^{t_{\mathsf{Int}}^{(i)}} = y^x \cdot (\prod_{j \neq i} \gamma^{t_{\mathsf{Int}}^{(j)}})^{-1}$, where y is the value given to $\mathcal{S}$ by $\mathcal{F}_{\mathsf{RSAGen}}$ during the Jacobi command and where $\mathcal{S}$ uses its internal values of x and $p^{(j)}, q^{(j)}$ to compute $t_{\mathsf{Int}}^{(j)}$ of the corrupt parties. $\mathcal{S}$ then broadcasts $\gamma^{t_{\mathsf{Int}}^{(j)}}$ on behalf of the honest parties.

5. When all parties call $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}.\mathsf{Open}([\![x, \ell_{\mathsf{Jac}}]\!])$, $\mathcal{S}$ simulates the opening and sends Consistency Check to $\mathcal{F}_{\mathsf{RSAGen}}$ on behalf of the corrupt parties.

6. To reply to $\mathcal{F}_{\mathsf{RSAGen}}$ about the abort, $\mathcal{S}$ checks whether $\prod_j \gamma^{t_{\mathsf{Int}}^{(j)}} = y^x$ using the values that were broadcast. If the equality fails, $\mathcal{S}$ sends Abort to $\mathcal{A}$ on behalf of the honest parties.

The values used by $\mathcal{S}$ in Step 2 are identically distributed to those in the protocol, so the distribution of the modified share $\gamma^{t_{\mathsf{Int}}^{(i)}}$ in Step 4 is statistically close, as measured by Lemma 1. As x is sampled at random identically, and the influence of $\mathcal{A}$ in the integer conversion of t is preserved, then the distribution of the broadcast of Step 4 is statistically close to the distribution of a real transcript.

Finally, the probability of abort when $\mathcal{A}$ behaves honestly remains the same, since Step 4 modifies the honest shares to be consistent with the y output. When $\mathcal{A}$ acts maliciously in the broadcast of $\gamma^{t_{\mathsf{Int}}^{(j)}}$, we claim that it has a negligible chance of making the equality hold, if it had already cheated in the Jacobi test. If it successfully makes the equality hold, this creates a difference between real and ideal world as $\mathcal{F}_{\mathsf{RSAGen}}$ would abort, since it received a modified y' from $\mathcal{S}$ but $\mathcal{S}$ would not abort. As we assume that $\mathcal{A}$ successfully cheated in the Jacobi test, we can assume that $\tilde{N} = (N-p-q+1)/4 \neq 0 \mod \phi(N)$ and that the adversary introduced an error Δ_j such that $y' = \gamma^{\tilde{N}+\Delta_j} = \pm 1 \mod N$. When the adversary cheats in the broadcast of the consistency check, the simulator computes $\gamma^{t_{\mathsf{Int}}+\Delta_t}$, where $\Delta_t \neq 0$ represents the error introduced by $\mathcal{A}$. Thus, for the equality to hold, $\mathcal{A}$ needs to commit to Δ_t during the broadcast, such that $\gamma^{x\tilde{N}+\Delta_t} = y^x = \gamma^{x\tilde{N}} \mod N$. Since the distribution of $x\tilde{N}$ is uniform with a

min-entropy of $2^{-\sec}$, because of the sampling of x, $\mathcal{A}$ has probability at most $2^{-\sec+1}$ of finding the correct Δ_t. We finally note that Δ_t cannot simply be the right value in the group of exponents with higher probability than this, because Rosser and Schoenfeld [34] showed that

$$\phi(N) > \frac{N}{e^{\gamma} \log\log N},$$

where γ is the Euler–Mascheroni constant. This gives us that $\log_2(\phi(N)) > \sec$, when $\log_2(N) > 2048$ and $\sec \sim 80$.

GCD Test:

1. When all parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{Rand2k}$ in Step 2, $\mathcal{S}$ samples a random $a < 2^{2\lambda+2\sigma}$ and receives the shares $(a^{(j)}, \ell_{\mathsf{gcd}})$ for each corrupt $P_j \in \mathcal{C}$ from $\mathcal{A}$. It then samples the remaining shares $(a^{(j)}, \ell_{\mathsf{gcd}})$ for honest $P_j \in \mathcal{H}$, such that $a = \mathsf{CRTrec}(a, \ell_{\mathsf{c}}) = \sum_{j=1}^{n}(a^{(j)}, \ell_{\mathsf{gcd}})$, and stores them.

2. When all parties call $\mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Mult}([\![z, \ell_{\mathsf{gcd}}]\!], [\![a, \ell_{\mathsf{gcd}}]\!], [\![(p+q-1), \ell_{\mathsf{gcd}}]\!])$, $\mathcal{S}$ internally executes the MPC multiplication.

3. When all parties call $\mathcal{F}_{\mathsf{AdvMPC-CRT}}.\mathsf{Rand2k}$ in Step 4, $\mathcal{S}$ samples a random $v < 2^{3\lambda+4\sigma}$ and receives the shares $(v^{(j)}, \ell_{\mathsf{gcd}})$ for each corrupt $P_j \in \mathcal{C}$ from $\mathcal{A}$. It then samples the remaining shares $(v^{(j)}, \ell_{\mathsf{gcd}})$ for honest $P_j \in \mathcal{H}$, such that $v = \mathsf{CRTrec}(v, \ell_{\mathsf{c}}) = \sum_{j=1}^{n}(v^{(j)}, \ell_{\mathsf{gcd}})$, and stores them.

4. Before opening, $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{RSAGen}}.\mathsf{GCD\ Test}$ and receives $b = \gcd(N, p+q-1)$. It then computes $b' = \gcd(N, a)$ and samples a new $\tilde{z}$ subject to the condition that $\gcd(N, \tilde{z}) = \max\{b, b'\}$. It finally replaces $\hat{z}$ by the new value $\hat{z} = \tilde{z} + v \cdot N$.

5. When all parties call $\mathcal{F}_{\mathsf{MPC-CRT}}.\mathsf{Open}([\![\hat{z}, \ell_{\mathsf{gcd}}]\!])$, $\mathcal{S}$ simulates the opening using the modified $\hat{z}$ and outputs Abort if $\mathcal{F}_{\mathsf{RSAGen}}$ output $\mathsf{Non\ Biprime}$.

If $\gcd(N, p+q-1)) = b$ then $b \mid z + v \cdot N$ and $b \mid z = a(p+q-1)$. Now, $\gcd(N, \hat{z})$ in the protocol can differ from $b = \gcd(N, p+q-1)$, if $\gcd(N, a) = b' > b$; thus, by sampling a random $\tilde{z}$ such that $\gcd(N, \tilde{z}) = \max\{b, b'\}$, the simulator remains consistent with both b and the probability that $\gcd(N, a) = b' > b$ occurs, since a is sampled identically. By adding a sufficiently random v to z any information about $p+q-1$ other than b is masked, therefore the distribution of the modified $\hat{z}$ output by the simulator is both statistically close to the distribution of $\hat{z}$ in the protocol, and consistent with the (N, p, q) values generated by $\mathcal{F}_{\mathsf{RSAGen}}$. □

Lemma 1. *In Step 5 of the Sampling Phase of Π_{RSAGen}, the distribution of each $p^{(j)}$ value is within statistical distance $(1-\epsilon)\epsilon M_{\mathsf{sample}}/S$ of uniform over*

$[2^\lambda, 2^{\lambda+\sigma})$, *where* $S = 2^{\lambda+\sigma} - 2^\lambda$ *is the size of the range and* $\epsilon = S/M_{\mathsf{sample}} - \lfloor S/M_{\mathsf{sample}} \rfloor \in [0,1)$ *is the decimal remainder in the division of the range size by* M_{sample}.

Proof. We can write $S = M_{\mathsf{sample}}(\lfloor S/M_{\mathsf{sample}} \rfloor + \epsilon)$ with $0 \leq \epsilon < 1$. When dividing the range of size S into blocks of size M_{sample}, the last block will not be complete (if M_{sample} does not divide S). When reducing the elements of $[2^\lambda, 2^{\lambda+\sigma})$ modulo M_{sample}, some residue classes will therefore be present one more time than others: those classes which have representatives in the included portion of the last block. Let X_1 denote the subset of $x \in [2^\lambda, 2^{\lambda+\sigma})$, whose residue class is more present, and let $X_2 = [2^\lambda, 2^{\lambda+\sigma}) \setminus X_1$ be those elements, whose residue class does not have a representative in the included portion.

Let $x \in [2^\lambda, 2^{\lambda+\sigma})$, by Euclidean division, x can be uniquely written as $x = aM_{\mathsf{sample}} + b$ with $a \in \mathbb{N}$ and $b \in [0, M_{\mathsf{sample}})$. As $p^{(j)}$ is computed as $p^{(j)} = p'^{(j)} + r^{(j)} \cdot M_{\mathsf{sample}}$ in Step 5 of the sampling phase, this implies:

$$\Pr\left[p^{(j)} = x\right] = \Pr\left[p'^{(j)} = b \wedge r^{(j)} = a\right] = \begin{cases} \frac{1}{M_{\mathsf{sample}}} \cdot \frac{1}{\lfloor S/M_{\mathsf{sample}} \rfloor + 1} & x \in X_1, \\ \frac{1}{M_{\mathsf{sample}}} \cdot \frac{1}{\lfloor S/M_{\mathsf{sample}} \rfloor} & x \in X_2, \end{cases}$$

as $p'^{(j)}$ is uniform in $[0, M_{\mathsf{sample}})$, because of $\mathcal{F}_{\mathsf{ABBWithErrors}}.\mathsf{Share}(p')$ and $r^{(j)}$ is uniform subject to the condition that $p^{(j)} \in [2^\lambda, 2^{\lambda+\sigma})$. Let P denote the above probability distribution. To compute the statistical distance between P and uniform, we compute the size of both X_1 and X_2, which is

$$|X_1| = \epsilon M_{\mathsf{sample}} \cdot \left(\left\lfloor \frac{S}{M_{\mathsf{sample}}} \right\rfloor + 1\right) \qquad \text{and} \qquad |X_2| = (1-\epsilon) M_{\mathsf{sample}} \cdot \left\lfloor \frac{S}{M_{\mathsf{sample}}} \right\rfloor.$$

This yields the following distance calculation:

$$\begin{aligned} \Delta(P, U) &= \frac{1}{2} \sum_{x \in [2^\lambda, 2^{\lambda+\sigma})} \left|\Pr[P = x] - \Pr[U = x]\right| \\ &= \frac{1}{2}\left(|X_1| \cdot \left|\Pr[P = x \mid x \in X_1] - \frac{1}{S}\right| + |X_2| \cdot \left|\Pr[P = x \mid x \in X_2] - \frac{1}{S}\right|\right) \\ &= \frac{1}{2}\left(\left|\epsilon - \frac{\epsilon M_{\mathsf{sample}}\left(\left\lfloor \frac{S}{M_{\mathsf{sample}}} \right\rfloor + 1\right)}{S}\right| + \left|(1-\epsilon) - \frac{(1-\epsilon) M_{\mathsf{sample}} \left\lfloor \frac{S}{M_{\mathsf{sample}}} \right\rfloor}{S}\right|\right) \\ &= \frac{1}{2}\left(\left|\frac{\epsilon M_{\mathsf{sample}}(\epsilon - 1)}{S}\right| + \left|\frac{(1-\epsilon)\epsilon M_{\mathsf{sample}}}{S}\right|\right) \\ &= \frac{(1-\epsilon)\epsilon M_{\mathsf{sample}}}{S}. \end{aligned}$$

□

5 Parameters and Efficiency Analysis

We generate biprimes of various bit-lengths, and hence security levels; namely $\lambda = \{1024, 1536, 2048\}$ as in the work of Chen et al. [7]. In the cases where a statistical security parameter σ needs to be considered, such as in the Sampling Phase, Jacobi test, masking and underlying MPC engines, we make sure to set $\sigma = 80$ to have a fair comparison with the analysis of Chen et al. [7], since they also used $\sigma = 80$, when measuring their concrete costs.

Given that our protocol requires several types of MPC engines, e.g., the ABBWithErrors, or the MPC-CRT, we use the MP-SPDZ framework [26] to get concrete communication costs for different adversary structures. In the case of dishonest majority, we instantiate ABBWithErrors using the semi-honest version of the MASCOT protocol [27], whereas for the malicious case, which we need for building the MPC-CRT, we use LowGear [28], with TopGear [3] as the underlying ZK proof. For the 16 parties case, we use the HighGear protocol with the TopGear ZK-proof, which is also implemented in MP-SPDZ. The reason for choosing HighGear over LowGear is that for HighGear communication scales better in the number of parties.

We also give concrete costs for RSA-Sieve in the semi-honest, dishonest majority model. The only difference with the malicious case is that MPC-CRT can be instantiated with a cheaper protocol and no zero-knowledge proofs. For this variant, we use the classical SPDZ triple generation with no ZK proofs [6, 11], for which we get concrete costs by running the *hemi* protocol in MP-SPDZ [26]. The results are given in Table 2 for the two party case, while in Table 3 we have results for the 16 party case, where we also compare them with the the protocol of Chen et al. [7]. As shown in Table 2, for two parties, our protocol is a factor of 3.3-3.9 more communication-efficient than the state-of-the-art [7] in the semi-honest case, and by a factor of 32-37 in the malicious case (ranging for different bit-lengths of the birprimes generated). For the 16 party case, the protocol of Chen et al. [7] outperforms ours by a factor of approximately 2 in the semi-honest case. We left the corresponding cell in Table 3 empty, to avoid confusion, as the rest of the improvement factors refer to our work outperfoming that of Chen et al. Then again, for the malicious case and for 16 parties, our protocol improves the communication cost over the state-of-the-art [7] by approximately 14-30 times.

Although Diogenes [8] works in a model much different than ours, namely the coordinator model, it is interesting to compare our estimates of the communication cost with theirs. An important feature of Diogenes [8] is that the communication cost per party, scales logarithmically in the number of parties, which is due to the message aggregation and relaying function of the coordina-

tor. For generating a 2048-bit modulus, running a semi-honestly secure version of their protocol, Diogenes [8] incurs a communication cost of 150MB per party, for 1024 parties. This makes the two-party instantiation of our protocol 3.6× more efficient than Diogenes [8], while the effect of better scalability becomes evident when compared to our 16-party instantiation, where Diogenes is 29× more communication efficient. Looking at the actively secure variant of Diogenes [8], again for the generation of a 2048-bit modulus with 1024 parties, the communication cost is 170MB per party. Thus, for the actively secure version of the protocols, Diogenes outperfoms both our two-party, and our 16-party instantation, by 3.8×, and 405×, respectively. Note, however, that the abovementioned communication cost estimates do not account for the coordinator, which receives and relays messages from all 1024 parties.

Scheme	[7]	Ours	[7]	Ours	[7]	Ours	Improve Factor Range
κ	1024	1024	1536	1536	2048	2048	
semi-honest (MB)	139	41.68	416	116.55	910	243.30	3.3-3.9×
malicious (GB)	20.81	0.64	43.42	1.188	74.52	1.99	32.5-37.4×

Table 2. Communication per party (two parties). For [7] the cost of the semi-honest protocol is based on the use of the OT extension of Keller et al. [27]. We consider this to be a fair comparison, as the sampling protocol is the major bottleneck and can be implemented using SilentOT. In our case the underlying MPC engine for sampling also used the same OT extension.

In the following, we give an example of how we compute the cost using $\lambda = 1024$, in the dishonest majority case with malicious security. The number of primes used in the distributed sieving is fixed to 130, as in the work of Chen et al. [7], to achieve the same number of Sample iterations. Note that the product of the first 130 primes is 1019 bits long. Frankel et al. [16] select $r^{(j)}$ in the sampling phase at random from $[0, 2^n/M_{\mathsf{sample}}]$ where n was the desired bit-length of p.

For λ being the bit-length of the candidate primes, we need to take their product in a space of double the size to avoid wrap around. Hence, M_{ℓ_c}, the product of primes in which the biprimes live, needs to be of length at least $2\lambda + 2\sigma$ bits, which results in $\ell_c = 18$ (i.e., we need 18 CRT components of 128 bits each). Similarly, we compute $\ell_{\mathsf{Jac}} = 21$ and $\ell_{\mathsf{gcd}} = 46$.

Scheme	[7]	Ours	[7]	Ours	[7]	Ours	Improve Factor Range
κ	1024	1024	1536	1536	2048	2048	
semi-honest (GB)	2.09	4.34	6.24	12.17	13.65	25.23	
malicious (GB)	1020	68.8	4734	153.20	8100	281.91	14.8-30.9×

Table 3. Communication per party malicious case (16 parties). For [7] the cost of the semi-honest protocol is based on the use of the OT extension of Keller et al. [27].

1. **Sampling phase.** The cost per semi-honest multiplication per party with $\mathsf{ABBWithErrors}$ is $(n-1)(128 \cdot k + k^2)$, where n is the number of parties and k is the field size [27]. Since the cost is quadratic in the field size, our $\mathsf{ABBWithErrors}$ will work over all the small primes composing M_{sample}.

   This brings the communication cost per triple at 17.027 kilobits with a total communication including the Beaver openings. The Input calls to $\mathsf{ABBWithErrors}$ in Step 4 amount to 0.264kbits. This makes step 3+4 having a cost of 17.556 kbits.

   The remaining cost here comes from the Input calls to $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$. This is instantiated using LowGear with TopGear as ZK proof, where the input tuple cost is 1.35 kbits for a 128-bit prime. This makes Step 6 in the Sampling phase amount to 48.67 kbits. One iteration of this phase has a total cost of 66.23 kbits.

2. **Combine.** The cost per multiplication triple using $\mathcal{F}_{\mathsf{MPC\text{-}CRT}}$ amounts to 12.862 kbits per party. This brings the cost of one execution of Step 2 to 231 kbits. The opening (Step 4) takes another 2.176 bits. One iteration of this step has a total cost of 233 kbits.

3. **Jacobi test.** The cost of this step is simply $\log_2(N) \cdot n$, which is $2n \cdot \lambda$ or about 4 kbits.

4. **Consistency check.** This step begins with a call to $\mathsf{LevelUp}$ from ℓ_{c} to ℓ_{Jac}, thus requiring $21 - 18 = 3$ inputs per party, as well as a multiplication and an opening in the additional CRT components. Summing up to a total of $8.1 + 38.6 + 0.4 = 47.1$ kbits. Next, parties call $\mathsf{Rand2k}$, which costs sec bit

generations in one of the CRT components, sec outputs in this CRT component, and sec inputs in all the CRT components per party, for a total cost of $\mathsf{sec}\cdot(12.862+20\cdot 2\cdot 1.35+0.128) = 8575$ kbits for $\mathsf{sec} = 128$. As before, the multiplication cost is simply $21\cdot 12.862 = 270.1$ kbits. The call to ConvInt requires a call to Rand2k and one opening, which amounts to $177084 + 2.7 = 177087$ kbits. Finally, the parties need to broadcast an element in $\mathbb{Z}_N$ and then open an element in all the ℓ_{Jac} CRT components, which requires communicating 4.8 kbits. One iteration of this phase has a total cost of 185984 kbits.

5. **GCD test.** Here again we start with a call to LevelUp, which costs 358.6 kbits, and then a call to Rand2k for a cost of 297387.2 kbits. Next is a multiplication on all the 46 CRT components for a total cost of 591.7 kbits. Second to last, we do a final call to Rand2k with bigger parameters, so the cost this time is 583999 kbits. Finally, we open $\hat{z}$ for 5.9 kbits. This phase thus requires a total of 882423 kbits.

   We present the detailed per-phase cost for 2 and 16 parties, and for $\lambda = \{1024, 1536, 2048\}$ in Table 4 for the malicious case, and in Table 5 for the semi-honest case.

Reducing the number of Input calls in generating bounded randomness. In the maBit protocol designed in [35], each random bit $[\![b]\!]$ produced in the main MPC engine producing randomness is later fed into the other MPC engines by every party calling Input command on a different sharing of b. By plugging their method directly into our Rand2k protocol, in order to generate n_B shared bits with n parties shared across ℓ engines will require $n_B \cdot n$ Input calls to each of the ℓ 128 bit prime MPC engines.

We can reduce the number of input calls by a factor of $\sim 128 - (\sigma + \log_2 m)$, where m is the batch size for generating maBits. The key insight is for parties to batch their bit shares instead of inputting them one by one. For example, if they want to batch 16 bit inputs at once, they can call $\mathcal{F}_{\mathsf{MPC}}.\mathsf{Input}(\sum_{k=0}^{15} 2^k b_k^{(j)} \bmod m_i)$, where $b_k^{(j)}$ would be party P_j's share of the k^{th} bit in the maBit protocol. We need to take into account now that the random linear combination at the end is done over slightly larger secrets (16 bits instead of a single one), so we need to increase the random coefficients by 16 bits in order for the security reduction to go through easily. The proof of this small optimization is relatively straightforward, since one can use this as an oracle to solve the MSSP problem described in [35] by simply scaling the random coefficients. To fit everything in a 128-bit prime MPC engine, we pack 16 inputs together, while maintaining a maBit batch of 2^{15} bits produced at once.

κ	1024		1536		2048	
n	2	16	2	16	2	16
per-phase cost for one instance (Megabits)						
Sieving	0.36	51.2	0.5	73.5	0.68	95.7
BP test	0.004	0.03	0.006	0.04	0.008	0.06
Check	45.63	4296	67.63	8071	92.4	13029
expected cost to sample a biprime (GBytes)						
E[Iterations]	3607	3607	7251	7251	11832	11832
E[Total]	0.64	68.8	1.18	153.2	1.99	281.91

Table 4. Communication per party: malicious case. The GCD test is included in E[Total], as that is an one-time cost. Check step happens σ times.

κ	1024		1536		2048	
n	2	16	2	16	2	16
per-phase cost for one instance (kilobits)						
Sieving	82.97	9391	118.10	13175	152.44	16784
BP test	4.096	32	6.144	49.152	8.192	65.536
expected cost to sample a biprime (megabytes)						
E[Iterations]	3607	3607	7251	7251	11832	11832
E[Total]	41.68	4346	116.55	12173	243.3	25230

Table 5. Communication per party: semi-honest case.

The honest majority case with active security. Since our protocol works with any actively secure protocol, where the secret reconstruction is linear, we can instantiate it with the most efficient protocols for MPC for large field arithmetic [9]. The cost analysis of such an instantiation can be seen in Table 6.

κ	1024	1536	2048
megabytes	105.26	222.99	401.452

Table 6. Communication per party: malicious honest majority case (3 parties).

References

1. Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 417–432. Springer, Heidelberg, August 2002.
2. Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 805–817. ACM Press, October 2016.
3. Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 274–302. Springer, Heidelberg, August 2019.
4. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.
5. Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 425–439. Springer, Heidelberg, August 1997.
6. Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. MOTION - A framework for mixed-protocol multi-party computation. Cryptology ePrint Archive, Report 2020/1137, 2020. `https://eprint.iacr.org/2020/1137`.
7. Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. Multiparty generation of an RSA modulus. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 64–93. Springer, Heidelberg, August 2020.
8. Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthuramakrishnan Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority. *IACR Cryptol. ePrint Arch.*, 2020:374, 2020.
9. Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 34–64. Springer, Heidelberg, August 2018.
10. Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 654–673. Springer, Heidelberg, September 2020.
11. 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.

12. Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed RSA key generation. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 183–200. Springer, Heidelberg, February 2010.

13. Yvo Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July/August 1994.

14. Yvo Desmedt. Some recent research aspects of threshold cryptography (invited lecture). In Eiji Okamoto, George I. Davida, and Masahiro Mambo, editors, *ISW'97*, volume 1396 of *LNCS*, pages 158–173. Springer, Heidelberg, September 1998.

15. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, August 1990.

16. Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In Brian A. Coan and Yehuda Afek, editors, *17th ACM PODC*, page 320. ACM, June / July 1998.

17. Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 331–361. Springer, Heidelberg, August 2018.

18. Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 225–255. Springer, Heidelberg, April / May 2017.

19. Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *17th ACM PODC*, pages 101–111. ACM, June / July 1998.

20. Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129. Springer, Heidelberg, August 1999.

21. Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 430–443. ACM Press, October 2016.

22. Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA key generation and threshold Paillier in the two-party setting. In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 313–331. Springer, Heidelberg, February / March 2012.

23. Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32(2):265–323, April 2019.

24. Lukas Helminger, Daniel Kales, Sebastian Ramacher, and Roman Walch. Multi-party revocation in sovrin: Performance through distributed trust. Cryptology ePrint Archive, Report 2020/724, 2020. `https://eprint.iacr.org/2020/724`.

25. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC press, 2020.

26. Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1575–1590, 2020.

27. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, October 2016.

28. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.

29. Michael Malkin, Thomas D. Wu, and Dan Boneh. Experimenting with shared generation of RSA keys. In *NDSS'99*. The Internet Society, February 1999.

30. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.

31. Guillaume Poupard and Jacques Stern. Generation of shared RSA keys by two parties. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 11–24. Springer, Heidelberg, October 1998.

32. Tal Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 89–104. Springer, Heidelberg, August 1998.

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

34. J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6(1):64 – 94, 1962.

35. Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for SPDZ. Cryptology ePrint Archive, Report 2019/1300, 2019. `https://eprint.iacr.org/2019/1300`.

36. Berry Schoenmakers and Pim Tuyls. Efficient binary conversion for Paillier encrypted values. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 522–537. Springer, Heidelberg, May / June 2006.

37. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

38. Nigel P. Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *LNCS*, pages 342–366. Springer, Heidelberg, December 2019.

39. Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

A Unauthenticated Arithmetic Black Box Protocols

$\Pi_{\mathsf{ABBWithErrors}}$

Initialize: The parties create the necessary preprocessing material for the online phase, receiving a number of multiplication triples $([a], [b], [c])$, and mask values $(r_i, [r_i])$.

Input: To share an input x_i, party P_i takes an available mask value $(r_i, [r_i])$ and does the following:

1. Broadcast $\epsilon \leftarrow x_i - r_i$.
2. The parties compute $[x_i] \leftarrow [r_i] + \epsilon$.

Multiply: On input $([x], [y])$ the parties do the following:

1. Take one multiplication triple $([a], [b], [c])$, compute $[\epsilon] \leftarrow [x] - [a], [\rho] \leftarrow [y] - [b]$, and open these shares to get ϵ, ρ, respectively.
2. Set $[z] \leftarrow [c] + \epsilon \cdot [b] + \rho \cdot [a] + \epsilon \cdot \rho$.

Share: On input $[x]$, each party P_i retrieves its own share $x^{(i)}$ of x from its local memory.

Fig. 14. Protocol for passively secure MPC adjusted for passive security from MASCOT [27].

$\Pi_{\mathsf{InputTuple}}$

Input: On Input (Input, P_j) from all parties do the following:

1. P_j samples $r \xleftarrow{\$} \mathbb{F}$
2. All parties output $[r]$ and P_j outputs r.

Fig. 15. Protocol for passively secure Input Tuples as presented in MASCOT [27].

$\Pi_{\mathsf{TripleGeneration}}$

Multiply:

1. Each party samples $a^{(i)} \xleftarrow{\$} \mathbb{F}, b^{(i)} \xleftarrow{\$} \mathbb{F}$.
2. Every ordered pair of parties P_i, P_j does the following:
   (a) Both parties call $\mathcal{F}_{\mathsf{ROT}}^{k,k}$, where P_i inputs $(a_1^{(i)}, \ldots, a_k^{(i)}) = \mathbf{g}^{-1}(a^{(i)}) \in \mathbb{F}_2^k$.
   (b) P_j receives $q_{0,h}^{(j,i)}, q_{1,h}^{(j,i)} \in \mathbb{F}$, and P_i receives $s_h^{(i,j)} = q_{a_h^{(i)},h}^{(j)}$, for $h = 1, \ldots, k$.
   (c) P_j sends $d_h^{(j,i)} = q_{0,h}^{(j,i)} - q_{1,h}^{(j,i)} + b^{(j)}, h \in [k]$.
   (d) P_i sets $t_h^{(i,j)} = s_h^{(i,j)} + a^{(i)} \cdot d_h^{(j,i)} = q_{0,h}^{(j,i)} + a_h^{(i)} \cdot b^{(j)}$, for $h = 1, \ldots k$. Set $q_h^{(j,i)} = q_{0,h}^{(j,i)}$.
   (e) P_i sets $\mathbf{c}_{i,j}^{(i)} = \langle \mathbf{g}, \mathbf{t} \rangle \in \mathbb{F}$, for $\mathbf{t}$ the above k-element vector.
   (f) P_j sets $\mathbf{c}_{i,j}^{(j)} = -\langle \mathbf{g}, \mathbf{q} \rangle \in \mathbb{F}$, for $\mathbf{q}$ the above k-element vector.
   (g) Now we have: $\mathbf{c}_{i,j}^{(i)} + \mathbf{c}_{i,j}^{(j)} = a^{(i)} \cdot b^{(j)} \in \mathbb{F}$
3. Each party P_i computes: $\mathbf{c}^{(i)} = a^{(i)} \cdot b^{(j)} + \sum_{j \neq i} (\mathbf{c}_{i,j}^{(i)} + \mathbf{c}_{j,i}^{(i)})$

Combine:

1. Sample $r, \hat{r} \xleftarrow{\$} \mathcal{F}_{\mathsf{Rand}}(\mathbb{F})$.
2. Each party P_i sets:
   (a) $a^{(i)} = \langle a^{(i)}, r \rangle, c^{(i)} = \langle c^{(i)}, r \rangle$, and
   (b) $\hat{a}^{(i)} = \langle a^{(i)}, \hat{r} \rangle, \hat{c}^{(i)} = \langle c^{(i)}, \hat{r} \rangle$

Output: $([a], [b], [c])$ as a valid triple.

Fig. 16. Protocol for Triple Generation adjusted for passive security from MASCOT [27].

B Standard functionalities

Functionality $\mathcal{F}_{\mathsf{MPC}}$

Let $[x]$ denote the identifier for a value x stored in the functionality. Let $A \subset \{1, \ldots, n\}$ denote the index set of the corrupted parties.
Input: Receive a value $x \in \mathbb{F}_p$ from some party and store x.
Mult$([x], [y])$**:** Compute $z = x \cdot y$ and store $[z]$.
Share$([x])$**:** For each $i \in A$ receive $x_i \in \mathbb{F}_p$ from the adversary. Sample uniform honest parties' shares $x_{j\, j \notin A}$ s.t. $\sum_{i=1}^{n} x_i = x$. Send x_i to P_i.
Random: Sample $r \xleftarrow{\$} \mathbb{F}_p$ and store $[r]$.
Sum$([x_1], \ldots, [x_k])$**:** Compute $x = x_1 + \cdots + x_k$ and store $[x]$.
GenBit(): Sample $b \xleftarrow{\$} \{0, 1\}$ and store $[b]$.
Open$([x])$**:** Send the value x to all parties.
OpenTo$([x], j)$**:** Send the value x to party P_j.

Fig. 17. Arithmetic MPC Functionality

Functionality $\mathcal{F}_{\mathsf{Broadcast}}$

1. Receive a value x from party P_j.
2. Send x to all parties $P_i, i \neq j$.

Fig. 18. Broadcast Functionality

Functionality $\mathcal{F}_{\mathsf{AgreeRandom}}$

1. Receive a value x_i from all parties P_i.
2. Compute $x = \sum x_i$.
3. Send x to all parties P_i.

Fig. 19. Functionality to Agree on a Common Public Value

Functionality $\mathcal{F}_{\mathsf{Rand}}$

Init: On input $(\mathsf{Init}, \mathsf{sid}, \mathbb{F})$ from all parties await for incoming messages.

Random: On input $(\mathsf{Random}, \mathsf{sid})$ from all parties sample $r \xleftarrow{\$} \mathcal{U}(\mathbb{F})$ and send it to $\mathcal{S}$. Wait for $\mathcal{S}$ reply: if message is OK then send r to all parties. If the message is Abort and send Abort to all parties and halt.

Fig. 20. Rand Functionality

Curriculum Vitae

Eleftheria Makri was born in Athens, Greece in 1987. She completed her M.Sc. in Information & Communication Systems Security at the University of the Aegean, School of Engineering (Karlovasi, Greece) in 2011. Between 2012 and 2014, she worked as a junior researcher at the University of Twente, the Netherlands, focusing on Privacy-Preserving Data Mining in Electronic Health Records. Since 2014, she functions as a lecturer and researcher in the area of Information Security, at Saxion University of Applied Sciences, the Netherlands. Concurrently, in 2017 she joined COSIC, KU Leuven, as a part-time research assistant, where she works towards the completion of her PhD degree, under the supervision of Prof. Bart Preneel, and Prof. Frederik Vercauteren.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
IMEC-COSIC
Kasteelpark Arenberg 10 box 2452
B-3001 Leuven