UNIVERSITY OF CALIFORNIA SAN DIEGO

Towards A Private New World: Algorithm, Protocol, and Hardware Co-Design for Large-Scale Secure Computation

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Computer Engineering)

by

M. Sadegh Riazi

Committee in charge:

Professor Farinaz Koushanfar, Chair
Professor Tara Javidi
Professor Andrew B. Kahng
Professor Ryan Kastner
Professor Truong Nguyen

2020

The dissertation of M. Sadegh Riazi is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California San Diego

2020

DEDICATION

To my beloved wife, my lovely sister, and my dear parents.

EPIGRAPH

"Any sufficiently advanced technology is indistinguishable from magic."

—Arthur C. Clarke

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my Ph.D. advisor, Professor Farinaz Koushanfar, for her genuine support, unconditional care, and invaluable guidance and help.

I would like to thank my committee members, Professor Tara Javidi, Professor Truong Nguyen, Professor Andrew B. Kahng, and Professor Ryan Kastner for their valuable suggestions and advice during my doctoral study. I want to express my gratitude to my mentors at Microsoft Research, Dr. Kristin Lauter and Dr. Kim Laine for their unequivocal support and guidance which enabled me to work on the cutting-edge technologies. I was fortunate to work with and learn from Professor Ahmad-Reza Sadeghi, Professor Ari Juels, Professor Thomas Schneider, Professor Tajana Rosing.

I am grateful to have worked with several brilliant researchers. In particular, I would like to thank Dr. Azalia Mirhoseini, Dr. Ebrahim Songhori, Dr. Ilya Razenshteyn, Dr. Hao Chen, Dr. Oxana Poburinnaya, Dr. Ilaria Chillotti, Dr. Rahul Chatterjee, Mohammad Samragh, Siam Hussain, Mojan Javaheripi, Mohsen Imani, Yeseong Kim, Yihe Dong, Dr. Wei Dai, and Blake Pelton.

Last but not least, I would also like to express my most sincere appreciation of my wife, my sister, my parents, and my dear friends for their emotional support. I would not be the same person without them.

The material in this dissertation is, in part, based on the following papers that are published or are under review.

Chapter 1 and 2, in part, has been published at (i) the Proceedings of 2018 ACM ASIA Conference on Computer and Communications Security (AsiaCCS) and appeared as: M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, Farinaz Koushanfar, “Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications”, and (ii) the Proceedings of 2019 USENIX Security Symposium and appeared as M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, Farinaz

Koushanfar, “XONN: XNOR-based Oblivious Deep Neural Network Inference”, and (iii) the Proceedings of 2019 IEEE Hardware Oriented Security and Trust (HOST) and appeared as: M. Sadegh Riazi, Mojan Javaheripi, Siam U Hussain, Farinaz Koushanfar, “MPCircuits: Optimized Circuit Generation for Secure Multi-Party Computation”, and (iv) the Proceedings of 2020 ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) and appears as: M. Sadegh Riazi, Kim Laine, Blake Pelton, Wei Dai, “HEAX: An Architecture for Computing on Encrypted Data”, and (v) the Proceedings of 2017 Design Automation Conference (DAC) and appeared as: M. Sadegh Riazi, Ebrahim M. Songhori, Farinaz Koushanfar, “PriSearch: Efficient Search on Private Data”, and (vi) the 2017 ACM Transactions on Embedded Computing Systems (TECS) and appeared as: M. Sadegh Riazi, Mohammad Samragh, Farinaz Koushanfar, “CAMsure: Secure ContentAddressable Memory for Approximate Search”, and a paper that has been submitted to the 2020 Design Automation Conference (DAC) as: M. Sadegh Riazi, Seyed Mohammad Chavoshian, Farinaz Koushanfar, “SynFi: Automatic Synthetic Fingerprint Generation”. The dissertation author was the primary author of this material.

Chapter 3, in part, has been published at the Proceedings of 2018 ACM ASIA Conference on Computer and Communications Security (AsiaCCS) and appeared as: M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, Farinaz Koushanfar, “Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications”. The dissertation author was the primary author of this material.

Chapter 4 and Appendix A, in part, has been published at the Proceedings of 2019 USENIX Security Symposium and appeared as M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, Farinaz Koushanfar, “XONN: XNOR-based Oblivious Deep Neural Network Inference”. The dissertation author was the primary author of this material.

Chapter 5, in part, has been published at the Proceedings of 2020 ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) and appears as: M. Sadegh Riazi, Kim Laine, Blake Pelton, Wei Dai, “HEAX: An Architecture for

Computing on Encrypted Data". The dissertation author was the primary author of this material.

Chapter 6, in part, has been published at the Proceedings of 2019 IEEE Hardware Oriented Security and Trust (HOST) and appeared as: M. Sadegh Riazi, Mojan Javaheripi, Siam U Hussain, Farinaz Koushanfar, "MPCircuits: Optimized Circuit Generation for Secure Multi-Party Computation". The dissertation author was the primary author of this material.

Chapter 7, in part, has been published at the Proceedings of 2017 Design Automation Conference (DAC) and appeared as: M. Sadegh Riazi, Ebrahim M. Songhori, Farinaz Koushanfar, "PriSearch: Efficient Search on Private Data". The dissertation author was the primary author of this material.

Chapter 8, in part, has been published at the 2017 ACM Transactions on Embedded Computing Systems (TECS) and appeared as: M. Sadegh Riazi, Mohammad Samragh, Farinaz Koushanfar, "CAMsure: Secure ContentAddressable Memory for Approximate Search". The dissertation author was the primary author of this material.

Chapter 9, in part, has been submitted to the 2020 Design Automation Conference (DAC) as: M. Sadegh Riazi, Seyed Mohammad Chavoshian, Farinaz Koushanfar, "SynFi: Automatic Synthetic Fingerprint Generation". The dissertation author was the primary author of this material.

This dissertation was supported, in parts, by the Office of Naval Research (ONR) N00014-17-1-2500, National Science Foundation (NSF)/Semiconductor Research Coporation (SRC) (1619261/2016-TS-2690), MURI (FA9550-14-1-0351), NSF GC@Scale CNS-1619261, and NSF Trust-Hub grant (CNS-1649423) grants.

VITA

2014	Bachelor of Science in Electrical Engineering, Sharif University of Technology, Tehran, Iran
2016	Master of Science in Computer Engineering, Rice University, Houston, Texas
2016-2020	Graduate Research Assistant, University of California San Diego, La Jolla, California
2020	Doctor of Philosophy in Electrical Engineering (Computer Engineering), University of California San Diego, La Jolla, California

PUBLICATIONS

M. Sadegh Riazi, Seyed Mohammad Chavoshian, Farinaz Koushanfar, "SynFi: Automatic Synthetic Fingerprint Generation", *ArXiv Preprint, 2020*.

M. Sadegh Riazi, Kim Laine, Blake Pelton, Wei Dai, "HEAX: An Architecture for Computing on Encrypted Data", *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2020.*

Siam U. Hussain, **M. Sadegh Riazi**, Farinaz Koushanfar, "The Fusion of Secure Function Evaluation and Logic Synthesis", *IEEE Security and Privacy (S&P) Magazine, 2020.*

Rahul Chatterjee, **M. Sadegh Riazi**, Tanmoy Chowdhury, Emanuela Marasco, Farinaz Koushanfar, Ari Juels, "Multisketches: Practical Secure Sketches Using Off-the-Shelf Biometric Matching Algorithms", *ACM Conference on Computer and Communications Security CCS, 2019.*

M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, Farinaz Koushanfar, "XONN: XNOR-based Oblivious Deep Neural Network Inference", *USENIX Security, 2019.*

Mohsen Imani, Yeseong Kim, **M. Sadegh Riazi**, John Messerly, Patric Liu, Farinaz Koushanfar, Tajana Rosing, "A Framework for Collaborative Learning in Secure High-Dimensional Space", *IEEE Cloud Computing (CLOUD), 2019.*

M. Sadegh Riazi, Mojan Javaheripi, Siam U Hussain, Farinaz Koushanfar, "MPCircuits: Optimized Circuit Generation for Secure Multi-Party Computation", *IEEE Hardware Oriented Security and Trust (HOST), 2019.*

Ebrahim M Songhori, **M. Sadegh Riazi**, Siam U Hussain, Ahmad-Reza Sadeghi, Farinaz Koushanfar, "ARM2GC: Succinct Garbled Processor for Secure Computation", *In Proceedings of Design Automation Conference (DAC), 2019.*

M. Sadegh Riazi, Bita Darvish Rouhani, Farinaz Koushanfar, "Deep Learning on Private Data", *IEEE Security and Privacy (S&P) Magazine, 2019.*

Hao Chen, Ilaria Chillotti, Oxana Poburinnaya, Ilya Razenshteyn, **M. Sadegh Riazi**, "SANNS: Scaling Up Secure Approximate k-Nearest Neighbors Search", *ArXiv Preprint arXiv:1904.02033, 2019.*

M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, Farinaz Koushanfar, "Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications", *ACM ASIA Conference on Computer and Communications Security (AsiaCCS), 2018.*

M. Sadegh Riazi and Farinaz Koushanfar, "Privacy-Preserving Deep Learning and Inference", *International Conference On Computer Aided Design (ICCAD), 2018.*

Hao Chen, Ilaria Chillotti, Oxana Poburinnaya, Ilya Razenshteyn, **M. Sadegh Riazi**, "Scaling Up Secure Nearest Neighbor Search", *Neural Information Processing Systems (NeurIPS) Workshop on Privacy-Preserving Machine Learning, 2018.*

Bita Darvish Rouhani, **M. Sadegh Riazi**, Farinaz Koushanfar, "DeepSecure: Scalable Provably-Secure Deep Learning", *In Proceedings of Design Automation Conference (DAC), 2018.*

Siam U. Hussain, **M. Sadegh Riazi**, Farinaz Koushanfar, "SHAIP: Secure Hamming Distance for Authentication of Intrinsic PUFs", *ACM Transactions on Design Automation of Electronic Systems (TODAES), 2018.*

M. Sadegh Riazi, Mohammad Samragh, Farinaz Koushanfar, "CAMsure: Secure Content-Addressable Memory for Approximate Search", *ACM Transactions on Embedded Computing Systems (TECS), 2017.*

M. Sadegh Riazi, Ebrahim M. Songhori, Farinaz Koushanfar, "PriSearch: Efficient Search on Private Data", *In Proceedings of Design Automation Conference (DAC), 2017.*

M. Sadegh Riazi, Ebrahim M Songhori, Ahmad-Reza Sadeghi, Thomas Schneider, Farinaz Koushanfar, "Toward Practical Secure Stable Matching", *In Proceedings of Privacy Enhancing Technologies (PoPETs), 2017.*

M. Sadegh Riazi, Beidi Chen, Anshumali Shrivastava, Dan Wallach, Farinaz Koushanfar, "Sub-Linear Privacy-Preserving Near-Neighbor Search", *ArXiv Preprint arXiv:1612.01835, 2016.*

M. Sadegh Riazi, Neeraj K. R. Dantu, L. N. Vinay Gattu, Farinaz Koushanfar, "GenMatch: Secure DNA Compatibility Testing", *IEEE Hardware Oriented Security and Trust (HOST), 2016.*

ABSTRACT OF THE DISSERTATION

Towards A Private New World: Algorithm, Protocol, and Hardware Co-Design for Large-Scale Secure Computation

by

M. Sadegh Riazi

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California San Diego, 2020

Professor Farinaz Koushanfar, Chair

Data privacy and security are among the grand challenges in the emerging era of massive data and collective intelligence. On the one hand, the rapid advances of several technologies, including artificial intelligence, are directly dependent on harnessing the full potential of data. On the other hand, such colossal collections of data inherently have sensitive information about individuals; explicit access to the data violates the privacy of content owners. While a number of elegant cryptographic solutions have been suggested for secure storage as well as secure transmission of data, the ability to compute on encrypted data at scale has remained a standing challenge. Secure computation is a set of developing technologies that enable processing on

the unintelligible version of the data. Secure computation can create a zero-trust platform where two or more individuals or organizations collaboratively compute on their shares of data without compromising data confidentiality. Computing on encrypted data removes several critical obstacles that prohibit scientific advances in which collaboration between distrusting parties is needed. Nevertheless, secure computation comes at the cost of significant computational overhead and higher communication between the pertinent parties. Currently, the high computational complexity prevents secure computation to be adopted in compute-intensive systems. This dissertation introduces several holistic algorithm-level, protocol-level, as well as hardware-level methodologies to enable the large-scale realization of the emerging secure computing and privacy technologies.

The key contributions of this dissertation are as follows:

- Introducing a novel secure computation framework in which several secure function evaluation protocols are integrated. The integration allows to choose a specific protocol to execute each unique operation based on the underlying mathematical characteristics of the protocol. The proposed methodology enables the secure execution of machine learning models 4-133$\times$ faster than the prior art.

- Designing a neural network transformation and a customized secure computation protocol for secure inference on deep neural networks. The transformation translates the contemporary neural network operations into several Boolean operations that can more efficiently be executed in secure computation protocols. The proposed transformation in conjunction with the customized protocol enable privacy-preserving medical diagnosis on four medical datasets for the first time.

- Design and end-to-end implementation of a new high-performance hardware architecture for computing on encrypted data. The proposed architecture outperforms high-end GPUs by more than 30$\times$ and modern CPUs by more than two orders of magnitude.

- Creating an efficient methodology based on hardware synthesis tools to produce compact Boolean circuit representation of a given function. The Boolean representation is optimized according to the cost function of secure computation protocols. The methodology reduces the computation and communication costs by up to $4\times$.

- Designing a new substring search algorithm customized for secure computation that does not require random access to the text. The proposed algorithm outperforms all state-of-the-art substring search algorithms when run within the secure computation protocol.

- Introducing the first secure content-addressable memory for approximate search. The design enables high-accuracy similarity-based approximate search while keeping the underlying data private without relying on a trusted server. The construction is the first to provide post-breach data confidentiality.

- Proposing a new methodology to create large-volume synthetic human fingerprints that are computationally indistinguishable from real fingerprints. The methodology enhances the security of any fingerprint-based authentication system.

Chapter 1

Introduction

Rapid advances in computing capabilities, together with exponential growth in the number of sensors, have led to the generation of an unprecedented volume of data. In addition, faster Internet connections have paved the way for ubiquitous use of online services. Thus, several privacy and security issues are emerging as a result of poor data management and insufficient security mechanisms. Data centers are increasingly prone to both internal and external hacks, and preserving consumers' data privacy is becoming more challenging every day. To this end, novel cryptographic solutions are needed to enhance the security of large-scale systems and the privacy of consumers.

Secure computation is a new set of technologies that enables *computation on the encrypted data*. In contrast to traditional encryption mechanisms that protect data in storage or transit, secure computation protects data during computation: closing the loop for data confidentiality. The security guarantees, however, come at the cost of several orders of magnitude computation and communication overhead, limiting the real-world deployment of this technology.

This thesis addresses several standing challenges related to secure computation. In particular, several algorithmic, protocol-level, and hardware-level methodologies are proposed that enable computing on encrypted data to become ubiquitous. In this section, I will review the

challenges as well as my proposed methodologies to address such problems.

1.1 Mixed-Protocol Secure Computation Framework for Machine Learning

Secure Function Evaluation (SFE) is one of the most influential achievements of modern cryptography. It allows two or more parties to evaluate a function on their inputs without disclosing the inputs to each other; that is, all inputs are kept private by the respective owners. In fact, SFE emulates a *trusted* third party which collects inputs from different parties and returns the result of the function to all (or a specific set of) parties. SFE has many applications in privacy-preserving biometric authentication [EFG+09], secure auctions [FPRSJ04], secure search [RSK17b], privacy-preserving machine learning [DGBL+16a], and data mining [LP00]. SFE was initially introduced in 1982 by Andrew Yao [Yao82] as the millionaires' problem in which two millionaires are interested to know which one has more money without disclosing their amount of wealth. Later on, the problem was formalized as a general question for any function. The two most prominent SFE protocols are Yao's Garbled Circuits (GC) [Yao86b] and the Goldreich-Micali-Wigderson (GMW) protocol [GMW87].

In theory, any function that can be represented as a Boolean circuit can be evaluated securely using GC or GMW protocols. For certain frequent operations such as multiplication, the Boolean circuit description can result in a substantial overhead. The number of Boolean gates in the circuit grows quadratically with respect to the bit-width of the operands. As a result, GC and GMW can often be too slow and hence are of limited practical value. In addition to generic secure computation protocols, i.e., GC and GMW, there exist solutions that enable a subset of operations while imposing less computational overhead. For instance, secret sharing based methodologies allow one to perform linear operations such as multiplication and addition on the encrypted data. Non-linear operations, however, are not supported by these methodologies.

I introduce Chameleon, a fast, modular, and hybrid (mixed-protocol) secure computation framework for machine learning tasks that utilizes GC, GMW, and additive secret sharing protocols and achieves unprecedented performance both in terms of runtime and communication between parties. The proposed methodology securely executes linear operations, such as multiplication and addition, on encrypted values using additive secret sharing. The computation on the additive shares takes place in the ring $\mathbb{Z}_{2^l}$, integers modulo a power-of-two number. As such, linear operations can be computed significantly faster than generic SFE protocols due to the inherent properties of the ring $\mathbb{Z}_{2^l}$.

Non-linear operations on the encrypted values will be computed using either of the generic protocols, i.e., GC or GMW. To switch the underlying protocol, one needs to change the secret types from additive shares to Yao and Boolean shares, and vice versa. Chameleon provides efficient mechanisms to realize this requirement and enable execution of different protocols interchangeably. In addition, Chameleon introduces a new secure Vector Dot Product (VDP) protocol that requires less computation and communication, both asymptotically and concretely, compared to prior standard protocols for VDP. Since VDP is usually the computational bottleneck of almost all machine learning models, Chameleon substantially reduces the secure evaluation of various machine learning models. To further reduce the communication between parties, certain compression and seed expansion techniques are incorporated. In a nutshell, Chameleon provides a new approach to securely execute any type of machine learning application with 4-133$\times$ faster runtime.

1.2 Neural Network Transforming for Secure Computation

The advent of big data and striking recent progress in artificial intelligence are fueling the impending industrial automation revolution. In particular, Deep Learning (DL) —a method based on learning Deep Neural Networks (DNNs) —is demonstrating a breakthrough in accuracy.

DL models outperform human cognition in a number of critical tasks such as speech and visual recognition, natural language processing, and medical data analysis. Given DL's superior performance, several technology companies are now developing or already providing DL as a service. They train their DL models on a large amount of (often) proprietary data on their own servers; then, an inference API is provided to the users who can send their data to the server and receive the analysis results on their queries. The notable shortcoming of this remote inference service is that the inputs are revealed to the cloud server, breaching the privacy of sensitive user data.

Oblivious inference is the task of running the DL model on the client's input without disclosing the input or the result to the server itself. Several solutions for oblivious inference have been proposed that utilize one or more cryptographic tools such as Homomorphic Encryption (HE) [BV14, BGV14], Garbled Circuits, GMW protocol [GMW87], and Secret Sharing (SS). Each of these cryptographic tools offer their own characteristics and trade-offs.

I introduce XONN, a novel end-to-end framework which provides a paradigm shift in the conceptual and practical realization of privacy-preserving interference on deep neural networks. The existing work has largely focused on the development of customized security protocols while using conventional fixed-point deep learning algorithms. XONN, for the first time, suggests leveraging the concept of the Binary Neural Networks (BNNs) in conjunction with the GC protocol. In BNNs, the weights and activations are restricted to binary (i.e, ± 1) values, substituting the costly multiplications with simple XNOR operations during the inference phase. The XNOR operation is known to be *free* in the GC protocol [KS08a]; therefore, performing oblivious inference on BNNs using GC results in the removal of costly multiplications. Using my approach, I show that oblivious inference on the standard DL benchmarks can be performed with minimal, if any, decrease in the prediction accuracy.

State-of-the-art approaches for oblivious inference require at least one round of interaction between client the server. This requirement, in turn, can substantially increase the runtime of oblivious inference in real-world settings where the network latency is high and unstable. One

of XONN's design principles is to have a *constant* rounds of interaction between client and the server regardless of the number of layers in the neural network. The removal of many interaction rounds paves the way to run very deep neural networks. For instance, XONN enables secure medical diagnosis on four medical datasets for the first time, i.e., breast cancer, diabetes, liver disease, and Malaria. Leveraging XONN, medical agencies can obtain AI-based medical diagnosis through third parties without disclosing patients's data: the private data remains encrypted at all times, even during the computation by the third party. Thus, medical agencies can enhance their diagnosis without violating patients' privacy.

1.3 High-Performance Hardware Architecture for Computing on Encrypted Data

Cloud computing has, in a short time, fundamentally changed the economics of computing. It allows businesses to quickly and efficiently scale to almost arbitrary-sized workloads; small organizations no longer need to own, secure, and maintain their own servers. However, cloud computing comes with significant risks that have been analyzed in the literature over the last decade (see [DWC10,HN08,SK11]). Specifically, many of these risks revolve around data security and privacy. For example, data in cloud storage might be exposed to both outsider and insider threats, and be prone to both intentional and unintentional misuse by the cloud provider. Recently, the European Union and the State of California have passed strong data privacy regulations. In this light, companies and organizations that possess highly private data are hesitant to migrate to the cloud, and cloud providers are facing increasing liability concerns.

Fully Homomorphic Encryption (FHE) provides provable security guarantees without any trust assumptions on the cloud provider, and it can be used to enable several secure and privacy-preserving cloud-based solutions. For instance, in the context of Machine Learning as a Service (MLaaS), FHE can be used to perform oblivious neural network inference [DGBL$^+$16b,DSC$^+$18]:

clients send the encrypted version of their data, the cloud server runs ML models on the encrypted queries, and returns the result to the clients. All intermediate and final results are encrypted and can only be decrypted by the clients. Perhaps, the most critical obstacle today to deploy FHE at large-scale is the enormous computation overhead compared to a plaintext counterpart in which data is not kept confidential.

The *ciphertext* in FHE schemes is a set (usually a pair) of polynomials with degree $n-1$ (vectors of n integers) modulo a big integer. One of the main challenges of designing an architecture for FHE is that homomorphic operations on ciphertexts involve computationally intensive modular arithmetic on big integers (with several hundred bits). These operations have convoluted data dependency among different parts of the computation, making it challenging to design a high-throughput architecture. Moreover, the degree of the underlying polynomials is enormous (in the order of several thousand). Storing the entire intermediate results on FPGA chip is prohibitive.

Prior work that propose customized hardware have taken one of these approaches: (i) Designing co-processors that only accelerate certain low-level ring operations [CRS16, CGRS14, WH13, CMO+14, DÖS14b, JGCM+15]; high-level operations are performed on the CPU-side, which makes the co-processors of limited practical use. (ii) Storing intermediate results on off-chip memory, which significantly degrades the performance [PNPM15] to the extent that it can be worse than naive software execution [RJV+18]. (iii) Designing a hardware for a fixed modest-sized parameter, e.g., $n = 2^{12}$ [RTJ+19]. However, encryption parameters determine the security-level and the maximum number of consecutive multiplications that one can perform on ciphertext, both of which are application-dependent. One of our primary design goals in HEAX is to have an architecture that can be readily used for a wide range of encryption parameters. In addition, we propose several techniques to efficiently store and access data from on-chip memory and minimize (or eliminate for some parameter sets) off-chip memory accesses.

I introduce HEAX (stands for Homomorphic Encryption Acceleration): a novel high-

performance hardware architecture for computing on (homomorphically) encrypted data. I design several optimized core computation blocks for fast modular arithmetic and introduce a new architecture for high-throughput Number-Theoretic Transform (NTT). NTT is a ubiquitous operation in FHE as well as many lattice-based cryptography systems. Efficient NTT engine directly improves the performance of these cryptosystems since NTT operation is usually the computational bottleneck. Building on top of the NTT module I design modules to perform high-level operations supported by FHE, thus accelerating any FHE-based privacy-preserving system. HEAX has been internally implemented and integrated with Microsoft Azure. Proof-of-concept realization of HEAX provides 36-81× higher computation capability compared to datacenter GPUs while consuming significantly less power. Compared to modern CPUs, HEAX has 164-268× higher performance.

1.4 Compact Circuit Representation for Secure Computation

Secure multi-party computation (MPC a.k.a., SMC) protocols provide a provably-secure method for multiple parties to jointly evaluate a function on their private inputs without disclosing the input values to each other. MPC protocols can be categorized into two main groups: protocols based on (i) the Goldreich-Micali-Wigderson paradigm [GMW87] and (ii) the Garbled-Circuit paradigm [Yao86b]. The original idea of two-party GC is later generalized for multi-party setting in the Beaver-Micali-Rogaway (BMR) protocol [BMR90]. Both paradigms require the underlying function to be represented as a *Boolean circuit*. The tools and methods for Boolean computations of two-party protocols are available, but they are not readily scalable or available for multiple parties. Present ad-hoc realization of secure multi-party tasks do not provide a holistic tool usable for a variety of other MPC applications.

I present the *first automated* methodology to generate Boolean circuits, customized for MPC protocols with state-of-the-art optimizations. Inspired by TinyGarble [SHS+15], the most

efficient Boolean circuit generator for the two-party setting, I leverage standard logic synthesis tools for this purpose. Note that two-party libraries such as TinyGarble cannot be used for the MPC problem since the synthesis technology libraries are not compatible with the MPC protocols. In addition, the order of logic computation (determined by the API and the Boolean netlist sorter) is radically different for two-party protocols. `MPCircuits` relies on designing new technology libraries for the logic synthesis tools customized for MPC protocols. My solution can be integrated with any cryptographic back-end engine for the MPC protocol, e.g., the realizations in [BELO16, CHK+12], to allow users to perform a holistic secure multi-party computation. The experimental results on five real-world applications, i.e., secure auction, voting, nearest-neighbor search, private set intersection, and stable matching demonstrates up to $4\times$ reduction of computational and communication overhead of the underlying MPC protocol due to a more optimized Boolean representation of the problem.

1.5 Efficient Search on Private Data

String search allows one to learn whether or not a query string is present within a usually much larger text. Substring search also has numerous applications in different fields, ranging from search engines to surveillance to genomic data processing. Many of these applications involve processing private information. It is ideal to perform the search while keeping this information secret. For example, string search on genomic data is used in personalized medicine and identifying criminals using FBI Combined DNA Index System (CODIS)[1]. Meanwhile, an individual's genomic data carries highly sensitive information about her and her family. Therefore, it is necessary to devise a solution for genomic string search with respect to certain security and privacy requirements. Another example is a government surveillance system that can search patterns or particular words (e.g., "bomb") in chat logs, text messages, and documents without

[1] https://www.fbi.gov/services/laboratory/biometric-analysis/codis

undermining citizens' privacy.

Substring search is a very well-studied problem in computer science, and many ingenious algorithms have been proposed to solve it efficiently. However, prior algorithms are optimized with respect to search within a *public* text, i.e., there exist a party where both query as well as the text are available in the unencrypted form (cleartext). In the privacy-preserving setting where both query and text hold sensitive data, it is crucial to keep query and text private to their respective owners. One can realize privacy-preserving substring search using generic SFE protocols by describing the substring algorithm as a Boolean circuit. Unfortunately, due to the radically different computation cost functions of atomic operations within secure computation compared to the cleartext execution, state-of-the-art substring search algorithms perform less efficiently compared to the brute-force algorithm. For instance, random access to the text has a constant cost independent from the text size assuming the text fully resides in the memory. In contrast, random access to the text is a very expensive operation in secure computation, thus, algorithms that need many random accesses will have high computational cost.

I introduce PriSearch, a novel privacy-preserving string search which does not need random access to the text while storing linear-size precomputed data (with respect to the keyword length). The computations required in PriSearch are mainly based on symmetric key encryption and are far less expensive compared to HE. I also design new synthesis libraries and leverage logic synthesis tools to automatically generate an optimized sequential circuit for PriSearch to achieve minimal cost in GC. Moreover, PriSearch is designed such that it can be easily extended to support multiple variants of string search.

1.6 Secure Content-Addressable Memory

Content Addressable Memories (CAMs) have conventionally been used for fast and efficient packet forwarding in Internet routers [PA01]. CAM can be viewed as hardware engine for

efficient lookup table search. Unlike traditional algorithmic solutions that require sequential access to memory elements, CAM provides a hard-wired architecture that searches the entire memory in a single clock cycle. Asides from their usage in networks, emerging methodologies have been proposed to use CAMs for the purpose of in-memory computing, aiming to improve the energy efficiency and performance of conventional processors [IKRR16, RGC+15, IPRR16, RIKR17]. This means that CAMs will be tightly coupled with the main processing elements of emerging computers, hence, their security is of great importance. Nevertheless, to the best of my knowledge, there has been no prior work to address the security of CAM. A naive solution for security is storing encrypted data in CAM. This approach is not feasible since the nature of semantically secure encryption mechanisms (e.g., AES) implies that the similarity of two elements is not preserved in the ciphertext domain. Hence, one cannot use the encrypted data to perform the approximate search using CAMs.

I propose CAMsure, a lightweight solution for securing emerging CAM technologies in the context of approximate search. Instead of writing the raw data on CAM, my approach stores distance-preserving hash embeddings to provide data privacy. The hashing scheme that is utilized in CAMsure is a variant of Locality Sensitive Hashing (LSH) [IM98a]. This family of hashing methods creates a randomized embedding of data while preserving the pairwise similarity. The methodology in CAMsure relies on the *curse of dimensionality* and introduces a new practical trade-off between the mutual information between hash bits and the accuracy of the search results. CAMsure is the first solution for CAMs that provides post-breach data security: even in the case of a data breach, the data stored in the CAM has no direct information about the original data.

1.7 Synthetic Human Fingerprints

Human fingerprints are frequently used in several applications for authentication and identification, ranging from smart doors to authorizing payments on cell phones. Evaluating the

performance and reliability of identification and verification fingerprint-based systems requires access to a large fingerprint database. However, in practice, obtaining a massive corpus of fingerprint images incurs a high cost. In many cases, the research groups that are developing fingerprint-based authentication systems, do not have access to a large publicly-available database. The performance of these systems is directly dependent on the quality and quantity of the available data.

In addition to the above obstacles, gathering fingerprint impressions of a large population of people raises severe privacy and security concerns. In case of a breach, the fingerprint of many users will be directly exposed to attackers and can be used to fool any other authentication systems that accept fingerprints. To this end, we study the task of generating *synthetic fingerprints* which can solve the challenges mentioned above. Synthetic fingerprints solve the availability concern as they can be generated for virtually any number of samples. Moreover, synthetic fingerprints are artificially generated; hence, they do not leak any information about real identities.

I present SynFi, a new comprehensive framework to automatically generate high-quality synthetic fingerprints at scale. My solution formulates the process of generating synthetic fingerprints as two parallel deep learning tasks based on Generative Adversarial Network (GAN) and Super-Resolution (SR) paradigm. In particular, SynFi formalizes and satisfies the following design goals to meet real-world expectations: (i) the generated samples should preserve the minutiae characteristics of fingerprints used for authentication systems, e.g., ridge structure, bifurcations, and ridge endings. (ii) An ideal system should be able to generate *full-finger impressions* as opposed to partial fingerprints. (iii) Synthetic fingerprints should be *computationally indistinguishable* from real impressions to be used as a means to extend the security of biometric storage systems. (iv) The system should be fully automated, requiring no manual feature engineering to have high *scalability*. SynFi satisfies all of the above requirements.

1.8 Impact

This dissertation affects several aspects of secure computation and introduces various methodologies, i.e., algorithms, protocols, and hardware, to enhance the efficiency of secure computation and reduce the communication overhead. The contributions of this dissertation can be applied to different applications ranging from healthcare to financial services to cloud computing. The open-sourced projects, as well as the Application Programming Interfaces (APIs), can be found at https://github.com/sadeghriazi.

1.9 Acknowledgements

Chapter 1, in part, has been published at (i) the Proceedings of 2018 ACM ASIA Conference on Computer and Communications Security (AsiaCCS) and appeared as: M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, Farinaz Koushanfar, "Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications", and (ii) the Proceedings of 2019 USENIX Security Symposium and appeared as M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, Farinaz Koushanfar, "XONN: XNOR-based Oblivious Deep Neural Network Inference", and (iii) the Proceedings of 2019 IEEE Hardware Oriented Security and Trust (HOST) and appeared as: M. Sadegh Riazi, Mojan Javaheripi, Siam U Hussain, Farinaz Koushanfar, "MPCircuits: Optimized Circuit Generation for Secure Multi-Party Computation", and (iv) the Proceedings of 2020 ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) and appears as: M. Sadegh Riazi, Kim Laine, Blake Pelton, Wei Dai, "HEAX: An Architecture for Computing on Encrypted Data", and (v) the Proceedings of 2017 Design Automation Conference (DAC) and appeared as: M. Sadegh Riazi, Ebrahim M. Songhori, Farinaz Koushanfar, "PriSearch: Efficient Search on Private Data", and (vi) the 2017 ACM Transactions on Embedded Computing Systems (TECS) and appeared as: M. Sadegh Riazi, Mohammad Samragh, Farinaz Koushanfar,

"CAMsure: Secure ContentAddressable Memory for Approximate Search", and a paper that has been submitted to the 2020 Design Automation Conference (DAC) as: M. Sadegh Riazi, Seyed Mohammad Chavoshian, Farinaz Koushanfar, "SynFi: Automatic Synthetic Fingerprint Generation". The dissertation author was the primary author of this material.

Chapter 2

Background

2.1 Cryptographic Protocols

In the following, we provide a concise overview of the basic protocols and concepts that are used in this thesis. Intermediate values are kept as secret shares of different types. We denote a share of value x, in type T, and held by party i as $\langle x \rangle_i^T$.

2.1.1 Oblivious Transfer Protocol

Oblivious Transfer (OT) is a building block for secure computation protocols. The OT protocol allows a receiving party $\mathcal{R}$ to obliviously select and receive a message from a set of messages that belong to a sending party $\mathcal{S}$, i.e., without letting $\mathcal{S}$ know which message was selected. In 1-out-of-2 OT, $\mathcal{S}$ has two l-bit messages x_0, x_1 and $\mathcal{R}$ has a bit b indicating the index of the desired message. After performing the protocol, $\mathcal{R}$ obtains x_b without learning anything about x_{1-b} and $\mathcal{S}$ learns no information about b. We denote n parallel 1-out-of-2 OTs on l-bit messages as OT_l^n.

The OT protocol requires costly public-key cryptography that significantly degrades the performance of secure computation. A number of methods have been proposed to perform a

large number of OTs using only a few public-key encryptions together with less costly symmetric key cryptography in a constant number of communication rounds [Bea96, IKNP03, ALSZ13]. Although the OT extension methods significantly reduce the cost compared to that of the original OT, the cost is still prohibitively large for complex secure computation that relies heavily on OT. However, with the presence of a semi-trusted third party, the parties can perform OT protocols with very low cryptographic computation cost as explained in Section 3.4.5.

2.1.2 Garbled Circuit Protocol

One of the most efficient solutions for generic secure two-party computation is Yao's Garbled Circuit (GC) protocol [Yao86b] that requires only a constant number of communication rounds. In the GC protocol, two parties, Alice and Bob, wish to compute a function $f(a,b)$ where a is Alice's private input and b is Bob's. The function $f(.,.)$ has to be represented as a Boolean circuit consisting of two-input gates, e.g., AND and XOR. For each wire w in the circuit, Alice generates and assigns two random k-bit strings, called *labels*, X_w^0 and X_w^1 representing 0 and 1 Boolean values where k is a security parameter, usually set to $k = 128$ [BHKR13]. Next, she encrypts the output labels of a gate using the two corresponding input labels as the encryption keys and creates a four-entry table called *garbled table* for each gate. The garbled table's rows are shuffled according to the point-and-permute technique [NPS99] where the four rows are permuted by using the Least Significant Bit (LSB) of the input labels as the permutation bits. Alice sends the garbled tables of all the gates in the circuit to Bob along with the labels corresponding to her input a. Bob also obliviously receives the labels for his inputs from Alice through OT. He then decrypts the garbled tables one by one to obtain the output labels of the circuit's output wires. Alice on the other hand has the mapping of the output labels to 0 and 1 Boolean values. They can learn the output of the function by sharing this information.

2.1.3 GMW Protocol

The Goldreich-Micali-Wigderson (GMW) protocol is an interactive secure multi-party computation protocol [GMW87, Gol09]. In the two-party GMW protocol, Alice and Bob compute $f(a,b)$ using secret-shared values, where a is Alice's private input and b is Bob's. Similar to the GC protocol, the function $f(.,.)$ has to be represented as a Boolean circuit. In GMW, the Boolean value of a wire in the circuit is shared between the parties: Alice has $\langle v \rangle_0^B$, Bob has $\langle v \rangle_1^B$, and the actual Boolean value is $v = \langle v \rangle_0^B \oplus \langle v \rangle_1^B$. Since the XOR operation is associative, the XOR gates in the circuit can be evaluated locally and without any communication between the parties. The secure evaluation of AND gates requires interaction and communication between the parties. The communication for the AND gates on the same level of the circuit can be done in parallel. Suppose an AND gate $x \wedge y = z$ (where $\wedge$ is the AND operation) where Alice has shares $\langle x \rangle_0^B$ and $\langle y \rangle_0^B$, Bob has shares $\langle x \rangle_1^B$ and $\langle y \rangle_1^B$, and they wish to obtain shares $\langle z \rangle_0^B$ and $\langle z \rangle_1^B$, respectively.

As shown in [DSZ15], the most efficient method for evaluating AND gates in the GMW protocol is based on Beaver's multiplication triples [Bea91]: Multiplication triples are random shared-secrets a, b, and c such that $\langle c \rangle_0^B \oplus \langle c \rangle_1^B = (\langle a \rangle_0^B \oplus \langle a \rangle_1^B) \wedge (\langle b \rangle_0^B \oplus \langle b \rangle_1^B)$. The triples can be generated offline using OTs (cf. [SZ13]) or by a semi-trusted third party (cf. Section 3.4.4). During the online phase, Alice and Bob use the triples to mask and exchange their inputs of the AND gate: $\langle d \rangle_i^B = \langle x \rangle_i^B \oplus \langle a \rangle_i^B$ and $\langle e \rangle_i^B = \langle y \rangle_i^B \oplus \langle b \rangle_i^B$. After that, both can reconstruct $d = \langle d \rangle_0^B \oplus \langle d \rangle_1^B$ and $e = \langle e \rangle_0^B \oplus \langle e \rangle_1^B$. This way, the output shares can be computed as $\langle z \rangle_0^B = (d \wedge e) \oplus (\langle b \rangle_0^B \wedge d) \oplus (\langle a \rangle_0^B \wedge e) \oplus \langle c \rangle_0^B$ and $\langle z \rangle_1^B = (\langle b \rangle_1^B \wedge d) \oplus (\langle a \rangle_1^B \wedge e) \oplus \langle c \rangle_1^B$.

2.1.4 BMR Protocol

For brevity, we only report proof-of-concept evaluations based on BMR [BMR90]. This protocol has two main phases: garbling and evaluation. In the first phase, all parties jointly create the *garbled* version of the circuit. In the second phase, each party receives partial information

from other parties and begins to evaluate the circuit locally. The garbling phase is usually the most costly stage in the protocol execution. However, since it is independent of the actual inputs from the participating parties, it can be pre-computed in advance.

Garbling. In this phase, all parties assign two random labels for every wire in the circuit, one for semantic value zero and one for semantic value one. We use notations consistent with [BELO16]: $k^i_{w,a} \in \{0,1\}^\kappa$ denotes random label of wire w for the semantic value $a \in \{0,1\}$ held by party P_i $i = 1...n$ where n is the total number of parties. κ is the security parameter and is usually set to 128. For each *gate*, parties encrypt output labels using F^2, a double-key pseudorandom function and use two input labels as keys. Consider gate g, with two inputs u and v, and output wire w. For example, in the case of an `AND` gate, output label for semantic value 1 ($k_{w,1}$) is encrypted using the two input labels of semantic value 1 ($k_{u,1}$ and $k_{v,1}$). Since there are four possible input combinations for any two-input Boolean gate, parties create four different encryptions of the correct output label and their corresponding input keys. The collection of all four encrypted values is called a *garbled table*. More precisely, for every $a,b \in \{0,1\}$, the output label for $c = g(a,b) \in \{0,1\}$ is encrypted as

$$\left\{ \left(\bigoplus_{i=1}^{n} F^2_{k^i_{u,a},\, k^i_{v,b}} (n_g \circ j) \oplus k^j_{w,g(a,b)} \right) \right\}_{j=1}^{n} \tag{2.1}$$

where n_g is the unique ID number for a gate and $\circ$ denotes concatenation operation. In order to mask the relationship between labels and actual semantic values, each party also assigns a *permutation bit* λ^i_w and sets $\lambda_w = \oplus_{i=1}^n \lambda^i_w$. All four encrypted values are permuted according to permutation bits.

Evaluation. In order to transfer the labels associated with the true value of an input wire, the *Oblivious Transfer* (OT) protocol is used. In OT, one party holds two (or multiple) messages m_i and another party holds the selection bit(s) b. At the end of the protocol, the receiver gets m_b and learns nothing about other message(s) while the sender learns nothing about the selection

bit(s). Given the collection of n keys for each input wire, all parties can decrypt one row of each garbled table (those connected to input gates) and generate the output keys of those gates. The evaluation process continues until output gates are reached. Therefore, the evaluation process can be computed locally once each party has the correct combination of all n keys for all input gates. Note that none of the intermediate values are revealed to any party. In fact, the semantic value of each wire is XOR-shared among all parties. All labels are unintelligible by themselves. At the end of the protocol, each party only sends her share of the output wires' labels such that everyone can locally compute the plaintext output result. Please see [BELO16] for more detailed explanation.

Free-XOR Optimization. Kolesnikov et al. [KS08b] proposed a method that eliminates the need for creating garbled tables for `XOR` gates, rendering them almost free of cost. To utilize this technique, each party P_i needs to create a one-time random number $R_i \in \{0,1\}^\kappa$. Same as before, $k^i_{w,0}$ is generated randomly but $k^i_{w,1}$ is set to $R_i \oplus k^i_{w,0}$ for every wire. Due to this correlation of labels, the output label of each `XOR` gate can be computed by XORing the two input labels without any communication between parties.

2.1.5 Additive Secret Sharing

In this protocol, a value is shared between two parties such that the addition of two secrets yields the true value. All operations are performed in the ring $\mathbb{Z}_{2^l}$ (integers modulo 2^l) where each number is represented as an l-bit integer. A ring is a set of numbers which is closed under addition and multiplication.

In order to additively share a secret x, a random number within the ring is selected, $r \in_R \mathbb{Z}_{2^l}$, and two shares are created as $\langle x \rangle^A_0 = r$ and $\langle x \rangle^A_1 = x - r \bmod 2^l$. A party that wants to share a secret sends one of the shares to the other party. To reconstruct a secret, one only needs to add the two shares $x = \langle x \rangle^A_0 + \langle x \rangle^A_1 \bmod 2^l$.

Addition, subtraction, and multiplication by a public constant value η ($z = x \circ \eta$) can be

done locally by the two parties without any communication: party i computes the share of the result as $\langle z \rangle_i^A = \langle x \rangle_i^A \circ \eta \bmod 2^l$, where $\circ$ denotes any of the aforementioned three operations. Adding/subtracting two secrets ($z = x \pm y$) also does not require any communication and can be realized as $\langle z \rangle_i^A = \langle x \rangle_i^A \pm \langle y \rangle_i^A \bmod 2^l$. Multiplying two secrets, however, requires one round of communication. Furthermore, the two parties need to have shares of precomputed Multiplication Triples (MTs). MTs refer to a set of three shared numbers such that $c = a \times b$. In the offline phase, party i receives $\langle a \rangle_i^A$, $\langle b \rangle_i^A$, and $\langle c \rangle_i^A$ (cf. Section 3.4.4). By having shares of an MT, multiplication is performed as follows:

1. Party i computes $\langle e \rangle_i^A = \langle x \rangle_i^A - \langle a \rangle_i^A$ and
   $\langle f \rangle_i^A = \langle y \rangle_i^A - \langle b \rangle_i^A$.

2. Both parties communicate to reconstruct e and f.

3. Party i computes its share of the multiplication as

$$\langle z \rangle_i^A = f \times \langle a \rangle_i^A + e \times \langle b \rangle_i^A + \langle c \rangle_i^A + i \times e \times f$$

For more complex operations, the function can be described as an Arithmetic circuit only consisting of addition and multiplication gates where in each step a single gate is processed accordingly.

2.2 CKKS Fully Homomorphic Encryption Scheme

The homomorphic property of FHE schemes enables computation on encrypted data without the access to the decryption key. For example, adding two ciphertexts results in a ciphertext that encrypts "summation of the corresponding plaintext values". Multiplication, however, is significantly more complicated. It increases the number of polynomials in the

resulting ciphertext; requiring an operation, called *relinearization*, to transform the ciphertext back to a pair of polynomials. In order to avoid the underlying plaintext values in the ciphertext to blow-up, an operation called *rescaling* is performed which divides the plaintext value by a constant number. To enable SIMD-style operations, an encoding step is performed by the client to embed many numbers in a single ciphertext. CKKS scheme supports *rotation* in which the numbers encoded in a ciphertext can be rotated.

Relinearization, rescaling, and rotation operations can be expressed as a unified operation called *Key Switching* (plus certain pre- and/or post-processing steps). Modular arithmetic operations can be computed more efficiently if ciphertext coefficients are represented in a Residue Number System (RNS). The full-RNS variant of the CKKS scheme was introduced in [CHK$^+$19]. Another orthogonal optimization based on NTT provides a more efficient polynomial multiplication. In what follows, we provide more background on CKKS.

Notation. Throughout the thesis, integers and real numbers are written in normal case, e.g. q. Polynomials and vectors are written in bold, e.g. $\mathbf{a}$. Vectors of polynomials and matrices are written in upper-case bold, e.g. $\mathbf{A}$. We use subscripts to denote the indices, e.g. $\mathbf{a}_i$ is the i-th polynomial or row of $\mathbf{A}$.

We assume that n is a power-of-two integer and define a polynomial ring $R = \mathbb{Z}[X]/(X^n+1)$ whose elements have degrees at most $n-1$ since $X^n = -1 \in R$. We write $R_q = R/qR$ for the residue ring of R modulo an integer q whose elements have coefficients in $[-\lfloor (q-1)/2 \rfloor, \lfloor q/2 \rfloor] \cap \mathbb{Z}$. In the actual computation, we represent coefficients in $[0, q-1] \cap \mathbb{Z}$. We denote by $\mathbf{u} \cdot \mathbf{v}$ the multiplication of two polynomials where the product is reduced modulo X^n+1 in R and further reduced modulo q in R_q. We denote by $\langle \mathbf{u}, \mathbf{v} \rangle$ the dot product of two vectors, which gives $\sum_i u_i \cdot v_i$. We denote by $\mathbf{u} \odot \mathbf{v}$ the coefficient-wise multiplication $(u_0 \cdot v_0, u_1 \cdot v_1, \ldots)$.

For a real number r, $\lfloor r \rceil$ denotes the nearest integer to r, and $\lfloor r \rfloor$ is the largest integer smaller than or equal to r. For an integer a, $[a]_p$ denotes the reduction of a modulo an integer p to

$[0,\ p-1] \cap \mathbb{Z}$. We use $\mathbf{a} \leftarrow \chi$ to denote sampling $\mathbf{a}$ according to distribution χ. For a finite set $\mathbb{S}$, $U(\mathbb{S})$ denotes the uniform distribution on $\mathbb{S}$.

Residue Number System (RNS). There is a well-known technique to achieve asymptotic/practical improvements in polynomial arithmetic over R_q with an RNS by choosing $q = \prod_{i=0}^{L} p_i$ where p_i's are pair-wise coprime integers, based on the ring isomorphism $R_q \mapsto \prod_{i=0}^{L} R_{p_i}$.

We denote the RNS representation of an element $\mathbf{a} \in R_q$ by $\overline{\mathbf{A}} = \left(\overline{\mathbf{a}}_i = [\mathbf{a}]_{p_i}\right)_{0 \le i \le L} \in \prod_{i=0}^{L} R_{p_i}$. The inverse mapping is defined based on the formula $\mathbf{a} = \sum_{i=0}^{L} \overline{\mathbf{a}}_i \pi_i \left[\pi_i^{-1}\right]_{p_i} \pmod{q}$, where $\pi_i = \frac{q}{p_i}$. Multiplications or additions in R_q, denoted by $\mathbf{c} = \texttt{Func}(\mathbf{a}, \mathbf{b})$, can be performed on their RNS representation: $\overline{\mathbf{c}}_i = \texttt{Func}(\overline{\mathbf{a}}_i, \overline{\mathbf{b}}_i)$ in R_{p_i} (in parallel), $i = 0, 1, \ldots, L$.

Gadget Decomposition. Let $\mathbf{g} \in \mathbb{Z}^d$ be a gadget vector and q an integer. The gadget decomposition, denoted by $\mathbf{g}^{-1}$, is a function from R_q to R^d which transforms an element $\mathbf{a} \in R_q$ into $\mathbf{A} \in R^d$, a vector of small polynomials such that $\mathbf{a} = \langle \mathbf{g}, \mathbf{A} \rangle \pmod{q}$. We integrate the RNS-friendly gadget decomposition from [BEHZ16, HPS19].

CKKS Subroutines. We briefly review relevant subroutines:

• $\underline{\texttt{CKKS.Setup}(\lambda)}$: For a security parameter λ, set a ring size n, a ciphertext modulus q, a special modulus p coprime to q, and a key distribution χ and an error distribution Ω over R.

• $\underline{\texttt{CKKS.SymEnc}(\mathbf{m},\ \texttt{sk})}$: Let $\mathbf{m} \in R$ be a given plaintext and $\texttt{sk} = \mathbf{s} \in R_{qp}$ be a secret key. Sample $\mathbf{a} \leftarrow U(R_{qp})$ and $\mathbf{e} \leftarrow \Omega$, compute $\mathbf{b} = -\mathbf{a} \cdot \mathbf{s} + \mathbf{e} \in R_{qp}$, and return the ciphertext $\texttt{ct} = (\mathbf{c}_0,\ \mathbf{c}_1) = (\mathbf{b},\ \mathbf{a})$.

• $\underline{\texttt{CKKS.KeyGen()}}$: Sample $\mathbf{s} \leftarrow \chi$. Return a secret key $\texttt{sk} = \mathbf{s}$ and a public key $\texttt{pk} = \texttt{SymEnc}(0,\ \texttt{sk})$.

• $\underline{\texttt{CKKS.KskGen}(\texttt{sk}', \texttt{sk})}$: Let $\texttt{sk} = \mathbf{s} \in R_{qp}$ be the generated secret key, $\texttt{sk}' = \mathbf{s}' \in R_{qp}$ be a different key, and a gadget vector $\mathbf{g} \in \mathbb{Z}^d$. Return a key switching key $\texttt{ksk} = (\mathbf{D}_0 \mid \mathbf{D}_1) \in R_{q_\ell p}^{(L+2)\times 2}$, where $(\mathbf{d}_{0,i},\ \mathbf{d}_{1,i}) \leftarrow \texttt{SymEnc}(g_i \cdot \mathbf{s}',\ \mathbf{s})$ for $i = 0,\ 1,\ \ldots,\ d-1$.

• $\underline{\texttt{CKKS.Add}(\texttt{ct}_0,\ \texttt{ct}_1)}$: Given ciphertexts $\texttt{ct}_0,\ \texttt{ct}_1 \in R_{q_\ell}^2$ encrypting $\texttt{pt}_0,\ \texttt{pt}_1 \in R$, generate $\texttt{ct}' = \texttt{ct}_0 + \texttt{ct}_1 \in R_{q_\ell}^2$ which is equivalent to the encryption of $\texttt{pt}_0 + \texttt{pt}_1 \in R$.

Algorithm 1 Optimized Modular Mult. | $\texttt{MulRed}(x,\ y,\ y',\ p)$

Input: $x, y \in \mathbb{Z}_p$, $p < 2^{w-2}$, and $y' = \lfloor y \cdot 2^w / p \rfloor$
Output: $z \leftarrow x \cdot y \pmod{p}$

1: $z \leftarrow x \cdot y \pmod{2^w}$ ▷ the lower word of the product
2: $t \leftarrow \lfloor x \cdot y' / 2^w \rfloor$ ▷ the upper word of the product
3: $z_\varepsilon \leftarrow t \cdot p \pmod{2^w}$ ▷ the lower word of the product
4: $z \leftarrow z - z_\varepsilon$ ▷ single-word subtraction
5: **if** $z \geq p$ **then**
6: $\quad z \leftarrow z - p$
7: **end if**

Two frequently used operations in homomorphic evaluation are modular reduction and modular multiplication:

• $\underline{\texttt{Mod}(x,\ p)}$: Used to perform modular reduction of a single-word or double-word integer [Bar87]. For a modulus p with at most w bits, given an integer $x \in \left[0,\ (p-1)^2\right]$, precompute $u = \left\lfloor 2^{2w}/p \right\rfloor$, and compute $z = x \pmod{p}$. $\texttt{Mod}(\mathbf{a}, p)$ performs $\texttt{Mod}(a_i,\ p)$ for all $i = 0,\ 1,\ \ldots,\ n-1$.

• $\underline{\texttt{MulRed}(x,\ y,\ y',\ p)}$: For w-bit words and a modulus $p < 2^{w-2}$, given $x,\ y \in \mathbb{Z}_p$ and precomputed $y' = \lfloor y \cdot 2^w / p \rfloor$, compute $x \cdot y \pmod{p}$ according to Algorithm 1.

2.3 Deep Neural Networks

The computational flow of a deep neural network is composed of multiple computational layers. The input to each layer is either a vector (i.e., $\mathbf{x} \in \mathbb{R}^n$) or a tensor (i.e., $\mathbf{X} \in \mathbb{R}^{m \times n \times k}$). The output of each layer serves as the input of the next layer. The input of the first layer is the raw data and the output of the last layer represents the network's prediction on the given data (i.e., inference result). In an image classification task, for instance, the raw image serves as the input to the first layer and the output of the last layer is a vector whose elements represent the probability that the image belongs to each category. Below we describe the functionality of neural network layers.

Linear Layers: Linear operations in neural networks are performed in Fully-Connected (FC) and Convolution (CONV) layers. The vector dot product (VDP) between two vectors $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$ is defined as follows:

$$\text{VDP}\ (\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{n} \mathbf{w}[i] \cdot \mathbf{x}[i]. \tag{2.2}$$

Both CONV and FC layers repeat VDP computation to generate outputs as we describe next.

A fully connected layer takes a vector $\mathbf{x} \in \mathbb{R}^n$ and generates the output $\mathbf{y} \in \mathbb{R}^m$ using a linear transformation:

$$\mathbf{y} = W \cdot \mathbf{x} + \mathbf{b}, \tag{2.3}$$

where $W \in \mathbb{R}^{m \times n}$ is the weight matrix and $\mathbf{b} \in \mathbb{R}^m$ is a bias vector. More precisely, the i-th output element is computed as $\mathbf{y}[\mathbf{i}] = \text{VDP}\ (W[i,:], \mathbf{x}) + \mathbf{b}[i]$.

A convolution layer is another form of linear transformation that operates on images. The input of a CONV layer is represented as multiple rectangular channels (2D images) of the same size: $\mathbf{X} \in \mathbb{R}^{h1 \times h2 \times c}$, where $h1$ and $h2$ are the dimensions of the image and c is the number of channels. The CONV layer maps the input image into an output image $\mathbf{Y} \in \mathbb{R}^{h1' \times h2' \times f}$. A CONV layer consists of a weight tensor $\mathbf{W} \in \mathbb{R}^{k \times k \times c \times f}$ and a bias vector $\mathbf{b} \in \mathbb{R}^f$. The i-th output channel in a CONV layer is computed by sliding the kernel $\mathbf{W}[:,:,:,i] \in \mathbb{R}^{k \times k \times c}$ over the input, computing the dot product between the kernel and the windowed input, and adding the bias term $\mathbf{b}[i]$ to the result.

Non-linear Activations: The output of linear transformations (i.e., CONV and FC) is usually fed to an activation layer, which applies an element-wise non-linear transformation to the vector/tensor and generates an output with the same dimensionality. In this thesis, we particularly utilize the Binary Activation (BA) function for hidden layers. BA maps the input operand to its sign value (i.e., $+1$ or -1).

Batch Normalization: A batch normalization (BN) layer is typically applied to the output

of linear layers to normalize the results. If a BN layer is applied to the output of a CONV layer, it multiplies all of the i-th channel's elements by a scalar $\boldsymbol{\gamma}[i]$ and adds a bias term $\boldsymbol{\beta}[i]$ to the resulting channel. If BN is applied to the output of an FC layer, it multiplies the i-th element of the vector by a scalar $\boldsymbol{\gamma}[i]$ and adds a bias term $\boldsymbol{\beta}[i]$ to the result.

Pooling: Pooling layers operate on image channels outputted by the CONV layers. A pooling layer slides a window on the image channels and aggregates the elements within the window into a single output element. Max-pooling and Average-pooling are two of the most common pooling operations in neural networks. Typically, pooling layers reduce the image size but do not affect the number of channels.

2.3.1 Generative Adversarial Networks

One family of deep learning models that has become very popular in recent years is Generative Adversarial Networks (GAN) [GPAM$^+$14]. Using GANs, one can estimate the distribution of a given dataset. The key idea in GANs is to train two different neural networks in parallel and make use of each of them to improve the other one. One of the networks, the *generative* network, tries to generate samples from the given data distribution. The other one, the *discriminator* network, is in charge of learning to distinguish the samples generated by the generative network from the real data samples. During the training phase, the feedback from the discriminator network is used to enhance the quality of the samples produced by the generative network.

After their introduction, GANs has been used in different areas and tasks. One of their most important applications is generating synthetic images. For example, [KLA19] generates artificial images of human faces and [PLWZ19] uses GANs to generate landscape images from doodles.

2.4 Human Fingerprints

Each fingerprint has a set of micro-features that can be used to uniquely identify the finger. *Minutiae* points correspond to the particular locations of fingerprint, e.g., ridge bifurcations or endings. Each Minutia is represented as a tuple of the location (x, y), orientation θ, and a quality factor q. Since different impressions of the same finger can result in drastically different minutiae tuples, fingerprint matching methodologies rely on a scale- and rotation-invariant algorithms to detect whether two sets of minutiae points belong to the same finger or not. An ideal synthetic fingerprint generator should produce impressions such that the distribution of minutiae points is not far from real ones.

2.5 Sub-String Search Algorithms

The set of all alphabet letters is denoted as Σ. For example, in genomic processing, the alphabet set is $\Sigma = \{$'A', 'C', 'G', 'T'$\}$. The size of the alphabet set is defined as $R = |\Sigma|$. Given a query Q and a text T, string search looks for a contiguous substring in T which is equal to Q. More precisely the output o is:

$$o = \begin{cases} 1, & \text{iff } \exists i \mid Q = T[i : i+m-1],\ 0 \leqslant i \leqslant n-m \\ 0, & \text{otherwise} \end{cases}$$

where $Q \in \{\Sigma\}^m$ and $T \in \{\Sigma\}^n$ (usually $m << n$). In a two-party privacy-preserving string search, Alice holds Q, and Bob holds T, and the goal is that one or both parties learn o while the contents of Q and T remain secret from the other party (m and n are public).

2.6 Content-Addressable Memory

Conventional Random Access Memories (RAMs) offer an address-based access to the data. During a read operation, a RAM takes an address and outputs the corresponding content. In RAM-based architectures, the data is physically separated from the processor. Current processors need to fetch the data from the memory, thus, they suffer from the overhead of data movement in different levels of the memory hierarchy.

Another trend of memory architectures offers content-based access to the data [KTFY03]. Unlike conventional RAMs, CAMs store a table of contents and identify the index of a row in the table that matches the query. In other words, CAMs take the content as the input and output the address in which the content is stored. This memory architecture searches the entire table in a single clock cycle, hence, it provides several benefits in terms of search speed and energy consumption.

Figure 2.1 presents a schematic view of the CAM architecture, which comprises an input buffer, a set of stored words, and an address encoder. Each stored word, called a match-line, is composed of multiple CAM cells that are responsible for holding bit values. In a search operation, the buffer distributes the input (search key) among all the words (rows) in the CAM. The output voltage of a cell is discharged if the stored value is not equal to the corresponding bit in the input query. If all cells within a row match the input query, the output voltage of the whole match-line remains high and triggers the address encoder. The address encoder then generates the matching index based on the triggered match line.

The focus of this work is to provide the security of CAM's contents. The functionality of CAM is to perform search operations; thus, we employ the CAM architecture to implement efficient lookup tables for secure near-neighbor search. Note that any lookup table can be implemented using conventional RAM architectures. Nevertheless, CAMs are superior compared to RAMs due to the following reasons:

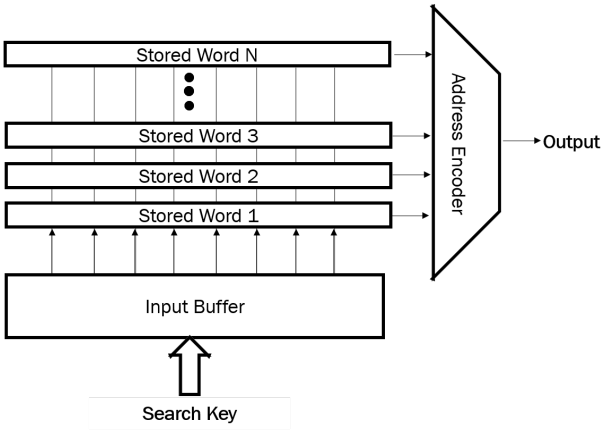


Figure 2.1: Schematic view of Content Addressable Memory (CAM).

- CAMs can search the entire table in a *single clock cycle* while RAMs perform *sequential* fetch and comparison operations.

- CAMs enable in-memory computation while RAMs require a processing unit which incurs additional delay and energy cost of data movement.

- CAMs reduce the response delay, improving the runtime of data-intensive applications.

CMOS-based CAMs [ACS03] and resistive CAMs [MHM$^+$09] have been proposed in the literature. CMOS-based CAMs are area-intensive and power consuming. For instance, a CMOS-based CAM consumes about 20$\times$ more power and requires 3.8$\times$ larger area compared to a RAM of the same storage capacity [GG10]. Alternatively, resistive cells deliver high-density and low-power CAMs [LMIC14, MKN$^+$11].

2.7 Approximate Search

In approximate search, the goal is to find all entries in a database that are *similar* to a given query q. Throughout this thesis, we use the terms *approximate search* and *near-neighbor search* interchangeably. We refer to each entry of the database as a *word*. The user's request is referred to as a *query*. Each data is represented as a D-dimensional vector whose elements are binary, integer, or real values depending on the underlying application. To quantify the

similarity, a metric (e.g., Jaccard, Cosine, or Euclidean) which is application-specific should be defined. For example, one of the most popular similarity metrics for web documents is the Jaccard similarity. In this case, each document is represented as a set. The Jaccard similarity for two sets $x, y \subseteq \Omega = \{1, 2, ..., |\Omega|\}$ is defined as

$$\mathcal{R} = \frac{|x \cap y|}{|x \cup y|}. \tag{2.4}$$

Another popular metric is the Cosine similarity. For two vectors $x, y \in \mathbb{R}^D$, the Cosine similarity can be computed as

$$\mathcal{C} = \frac{x^T y}{||x||_2 \cdot ||y||_2}, \tag{2.5}$$

where $||.||_2$ denotes norm-2 of a vector.

The output of the search is the indices of words in the database whose similarities are more than a predefined threshold (T_S). More precisely, we are looking for

$$Search(q) = \{i \mid Similarity(d_i, q) > t\},$$

where d_i is the i^{th} word in the database. Without loss of generality, the similarity measurement can always be normalized to a value between zero (no similarity) and one (identical). The approximate search is called *secure* if the database holder outputs indices i without inferring any information about the query q and words d_i.

LSH is a popular approximate search method that creates a probabilistic embedding (hash) of a data with the following property: if two inputs are similar in the input domain, their hashes have a higher probability of collision. More precisely,

$$Probability\{h(x) = h(y)\} \; = \; f(Similarity(x, y)) \tag{2.6}$$

where $h(.)$ is the hash function and $f(.)$ is a monotonically increasing function. There exist

variants of LSH that preserve different similarity metrics. For instance, MinHash preserves Jaccard similarity and SimHash preserves Cosine similarity. We briefly describe each of these hash functions while the reader can refer to [RCS$^+$16] for more detailed explanation.

Jaccard Similarity and MinHash

As we discussed in Section 2.7, Jaccard similarity is defined over sets. To compute the hash value, a random permutation $\pi : \Omega \rightarrow \Omega$ is applied on the input set. The hash is the minimum value of the permuted set; more precisely, $h_{min}(x) = min(\pi(x))$. For instance, consider the set $x = \{2,4,5\} \subset \Omega = \{1,2,3,4,5\}$ and the random permutation

$$\pi : 1 \rightarrow 4, 2 \rightarrow 1, 3 \rightarrow 5, 4 \rightarrow 2, 5 \rightarrow 3$$

that maps the set x to $\pi(x) = \{1,2,3\}$, hence, $h_{min}(x) = 1$. It has been shown [RCS$^+$16] that MinHash is a valid LSH since

$$Probability\{h_{min}(x) = h_{min}(y)\} \ = \ \mathcal{R},$$

where $\mathcal{R}$ is the Jaccard similarity defined in Equation 2.4 and the function $f(.)$ in Equation 2.6 is equal to $f(\alpha) = \alpha$ which is a monotonically increasing function.

Alternatively, each set can be represented as a binary vector of length $|\Omega|$ where the j^{th} binary value indicates whether j is a member of the set (value one) or not (value zero). Therefore, MinHash can be viewed as a function over binary vectors of length $D = |\Omega|$, $h_{min} : \{0,1\}^D \rightarrow \mathbb{N}$. In order to consume less storage space, it is preferable to generate 1-bit LSH ($h_{min}^{1-bit} : \{0,1\}^D \rightarrow \{0,1\}$) by applying a universal hash function to the output of MinHash (see [RCS$^+$16] for more information about the universal hash function). The collision probability of $h_{min}^{1-bit}(x)$ with $h_{min}^{1-bit}(y)$ is shown to be $1 + \frac{\mathcal{R}}{2}$ [RCS$^+$16]. One needs to perform the hash evaluation l times with l different random permutations to generate an l-bit LSH embedding.

Cosine Similarity and SimHash

SimHash is a popular LSH method that preserves Cosine similarity and is based on Signed Random Projections (SRP) [LRU14] which can be computed as follows: first, a random D-dimensional row vector w is generated where vector components are drawn from i.i.d normal distributions, $w_i \sim N(0,1)$. The 1-bit SimHash is generated by computing the sign of the projection of input vector to the randomly generated vector w, $h_{sim}(x) = sign(w \cdot x^T)$. In fact, we have $h_{sim} : \mathbb{R}^D \rightarrow \{0,1\}$. It is not difficult to show the following

$$Probability\{h_{sim}(x) = h_{sim}(y)\} \ = \ 1 - \frac{\theta}{\pi},$$

where $\theta = \cos^{-1}(\frac{x^T y}{||x||_2 \cdot ||y||_2})$. Comparing this with Equation 2.6, the function $f(.)$ can be formulated as $f(\alpha) = 1 - \frac{\cos^{-1}(\alpha)}{\pi}$. In order to generate an l-bit LSH embedding, we need to compute 1-bit SimHashes l times with l different randomly sampled w vectors.

2.8 Acknowledgements

This chapter, in part, has been published at (i) the Proceedings of 2018 ACM ASIA Conference on Computer and Communications Security (AsiaCCS) and appeared as: M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, Farinaz Koushanfar, "Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications", and (ii) the Proceedings of 2019 USENIX Security Symposium and appeared as M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, Farinaz Koushanfar, "XONN: XNOR-based Oblivious Deep Neural Network Inference", and (iii) the Proceedings of 2019 IEEE Hardware Oriented Security and Trust (HOST) and appeared as: M. Sadegh Riazi, Mojan Javaheripi, Siam U Hussain, Farinaz Koushanfar, "MPCircuits: Optimized Circuit Generation for Secure Multi-Party Computation", and (iv) the Proceedings of 2020 ACM

International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) and appears as: M. Sadegh Riazi, Kim Laine, Blake Pelton, Wei Dai, "HEAX: An Architecture for Computing on Encrypted Data", and (v) the Proceedings of 2017 Design Automation Conference (DAC) and appeared as: M. Sadegh Riazi, Ebrahim M. Songhori, Farinaz Koushanfar, "PriSearch: Efficient Search on Private Data", and (vi) the 2017 ACM Transactions on Embedded Computing Systems (TECS) and appeared as: M. Sadegh Riazi, Mohammad Samragh, Farinaz Koushanfar, "CAMsure: Secure ContentAddressable Memory for Approximate Search", and a paper that has been submitted to the 2020 Design Automation Conference (DAC) as: M. Sadegh Riazi, Seyed Mohammad Chavoshian, Farinaz Koushanfar, "SynFi: Automatic Synthetic Fingerprint Generation". The dissertation author was the primary author of this material.

Chapter 3

Chameleon: Mixed-Protocol Secure Computation Framework for Machine Learning

We present Chameleon, a novel hybrid (mixed-protocol) framework for secure function evaluation (SFE) which enables two parties to jointly compute a function without disclosing their private inputs. Chameleon combines the best aspects of generic SFE protocols with the ones that are based upon additive secret sharing. In particular, the framework performs linear operations in the ring $\mathbb{Z}_{2^l}$ using additively secret shared values and nonlinear operations using Yao's Garbled Circuits or the Goldreich-Micali-Wigderson protocol. Chameleon departs from the common assumption of additive or linear secret sharing models where three or more parties need to communicate in the online phase: the framework allows two parties with private inputs to communicate in the online phase under the assumption of a third node generating correlated randomness in an offline phase. Almost all of the heavy cryptographic operations are precomputed in an offline phase which substantially reduces the communication overhead. Chameleon is both scalable and significantly more efficient than the ABY framework (NDSS'15) it is based on. Our

framework supports signed fixed-point numbers. In particular, Chameleon's vector dot product of signed fixed-point numbers improves the efficiency of mining and classification of encrypted data for algorithms based upon heavy matrix multiplications. Our evaluation of Chameleon on a 5 layer convolutional deep neural network shows 133x and 4.2x faster executions than Microsoft CryptoNets (ICML'16) and MiniONN (CCS'17), respectively.

3.1 Introduction

Secure computation frameworks can be categorized based on the offline/online runtime, the number of computing nodes (two-party or multi-party), offline/online communication, the set of supported instructions, and the programming language which describes the functionality. These frameworks accept the description of the function as either (i) their own customized languages [MNPS04, MGC+16], (ii) high-level languages such as `C/C++` [HFKV12] or `Java` [HEKM11, LWN+15], or (iii) Hardware Description Languages (HDLs) [SHS+15, DDK+15].

A number of SFE compilers have been designed for translating a program written in a high level language to low-level code [MNPS04, HKS+10, MGC+16] [BK15b, BHWK16, BKJK16]. The low-level code is supported by other SFE frameworks which serve as a backbone for executing the cryptographic protocols. In addition to generic SFE protocols, additive/linear *secret sharing* enables secure computation of linear operations such as multiplication, addition, and subtraction. In general, each framework introduces a set of trade-offs. The frameworks based on secret-sharing require three (or more) computing nodes which operate on distributed shares of variables in parallel and require multiple rounds of communication between nodes to compute an operation on shares of two secret values.

One of the most efficient secure computation frameworks is Sharemind [BLW08] which is based on Additive Secret Sharing (A-SS) over the specific ring $\mathbb{Z}_{2^{32}}$. All operations are performed

by three computing nodes. Sharemind is secure against honest-but-curious (semi-honest) nodes which are assumed to follow the protocol but they cannot infer any information about the input and intermediate results as long as the majority of nodes are not corrupted. We consider the same adversary model in this thesis. Securely computing each operation in Sharemind needs multiple communication rounds between all three nodes which makes the framework relatively slow in the Internet setting. Computation based on additive shares in the ring $\mathbb{Z}_{2^l}$ enables very efficient and fast linear operations such as Multiplication (MULT), Addition (ADD), and Subtraction (SUB). However, operations such as Comparison (CMP) and Equality test (EQ) are not as efficient and *non-linear* operations cannot easily be realized in the ring $\mathbb{Z}_{2^l}$.

We introduce Chameleon, a fast, modular, and hybrid (mixed-protocol) secure two-party computation framework that utilizes GC, GMW, and additive secret sharing protocols and achieves unprecedented performance both in terms of run-time and communication between parties. The analogy comes from the fact that similar to a chameleon that changes its color to match the color of the environment, our framework allows changing the executing SFE protocol based on the run-time operation. The main design goal behind Chameleon is to create a framework that combines the advantages of the previous secure computation methodologies.

The idea of a mixed-protocol solution was first introduced in [BPSW07] which combines GC with Homomorphic Encryption (HE). HE enables to perform MULT and ADD operations on encrypted values without actually knowing the unencrypted data. The TASTY framework [HKS+10] enables automatic generation of protocols based on GC and HE. However, due to the high computational cost of HE and costly conversion between HE and GC, they achieve only a marginal improvement compared to the single protocol execution model [KSS14].

Our framework Chameleon is based on ABY [DSZ15] which implements a hybrid of additive SS, GMW, and GC for efficient realization of SFE. However, we overcome two major limitations, thereby improving efficiency, scalability, and practicality: The ABY model relies on oblivious transfers for precomputing arithmetic triples which we replace by more efficient

protocols using a Semi-honest Third Party (STP). The STP can be a separate computing node or it can be implemented based on a smartcard [DSZ14] or Intel Software Guard Extensions (SGX) [BBB+17]. Therefore, the online phase of Chameleon only involves two parties that have private inputs. Additionally, we extend ABY to handle signed fixed-point numbers which is needed in many deep learning applications, but not provided by ABY and other state-of-the-art secure computation frameworks such as TASTY.

Chameleon supports 16, 32, and 64 bit signed fixed-point numbers. The number of bits assigned to the fraction and integral part can also be tuned according to the application. The input programs to Chameleon can be described in the high-level language `C++`. The framework itself is also written in `C++` which delivers fast execution. In addition to a rich library of pre-defined functions, the user can simply add any function description as a Boolean circuit or a `C/C++` program to our framework and use them seamlessly.

Machine Learning on Private Data Using Chameleon. Chameleon's efficiency helps us to address a major problem in contemporary secure machine learning on private data. Matrix multiplication (or equivalently, vector dot product computation) is one of the most frequent and essential building blocks for many machine learning algorithms and applications. Therefore, in addition to scalability and efficiency described earlier, we design an efficient secure vector dot product protocol based on the Du-Atallah multiplication protocol [DA01] that has very fast execution and low communication between the two parties. We address secure Deep Learning (DL) which is a sophisticated task with increasing attraction. We also provide privacy-preserving classification based on Support Vector Machines (SVMs).

The fact that many pioneering technology companies have started to provide Machine Learning as a Service (MLaaS[1,2,3]) proves the importance of DL. Deep and Convolutional Neural Networks (DNNs/CNNs) have attracted many machine learning practitioners due to their capabil-

[1]Amazon AWS AI (`https://aws.amazon.com/amazon-ai/`)

[2]Google Cloud Machine Learning Engine (`https://cloud.google.com/ml-engine/`)

[3]Microsoft Azure Machine Learning Services (`https://azure.microsoft.com/services/machine-learning-services/`)

ities and high classification accuracy. In MLaaS, clients provide their inputs to the cloud servers and receive the corresponding results. However, the privacy of clients' data is an important driving factor. To that end, Microsoft Research has announced CryptoNets [DGBL$^+$16a]. CryptoNets is an HE-based methodology that allows secure evaluation (inference) of encrypted queries over *already trained* neural networks on cloud servers: queries from the clients can be classified securely by the trained neural network model on a cloud server without inferring any information about the query or the result.

Our Contributions. Our main contributions are as follows:

- We introduce Chameleon, a novel mixed SFE framework based on ABY [DSZ15] which brings benefits in terms of efficiency, scalability, and practicality by integrating signed fixed-point arithmetic, STP-based protocols for precomputing OTs and generating arithmetic and Boolean multiplication triples, and an optimized STP-based vector dot product protocol for vector/matrix multiplications.

- We provide detailed performance evaluation results of Chameleon compared to state-of-the-art frameworks. Compared to ABY, Chameleon requires up to 321$\times$ and 256$\times$ less communication for generating arithmetic and Boolean multiplication triples, respectively.

- We present a proof-of-concept implementation and experimental results on deep and convolutional neural networks. Comparing to the state-of-the-art Microsoft CryptoNets [DGBL$^+$16a], we achieve a 133x performance improvement. Comparing to the recent work of [LJLA17a], we achieve a 4.2x performance improvement using a comparable configuration.

3.2 Related Work

Chameleon is essentially a two-party framework that uses a Semi-honest Third Party (STP) to generate correlated randomness in the offline phase. In the following, we review the use of third parties in secure computation as well as other secure two-party and multi-party computation frameworks.

Third Party-based Secure Computation. Regarding the involvement of a third party in secure two-party computation, there have been several works that consider an outsourcing or *server-aided* scenario, where the resources of one or more *untrusted* servers are employed to achieve sub-linear work in the circuit size of a function, even workload distribution, and output fairness. Realizing such a scenario can be done by either employing fully-homomorphic encryption (e.g., [AJLA$^+$12]) or extending Yao's garbled circuit protocol (e.g., [KMR12]). Another important motivation for server-aided SFE is to address the issue of low-powered mobile devices, as done in [CMTB13,CLT14,DSZ14,CMTB15,MOR16,CMTB16]. Furthermore, server-aided secure computation can be used to achieve stronger security against active adversaries [HS12].

The secure computation framework of [Hua12, Chapter 6] also utilizes correlated randomness. Beyond passive security and one STP, this framework also covers active security and multiple STPs.

GC-based Frameworks. The first implementation of the GC protocol is Fairplay [MNPS04] that allows users to write the program in a high-level language called Secure Function Definition Language (SFDL) which is translated into a Boolean circuit. FairplayMP [BDNP08] is the extension of Fairplay to the multiparty setting. FastGC [HEKM11] reduces the running time and memory requirements of the GC execution by using pipelining. TinyGarble [SHS$^+$15, HRK20] is one of the recent GC frameworks that proposes to generate compact and efficient Boolean circuits using industrial logic synthesis tools. TinyGarble also

supports sequential circuits (cyclic graph representation of circuits) in addition to traditional combinational circuits (acyclic graph representation). ObliVM [LWN+15] provides a domain-specific programming language and a secure computation framework that facilitates the development process. Frigate [MGC+16] is a validated compiler and circuit interpreter for secure computation. Also, the authors of [MGC+16] test and validate several secure computation compilers and report the corresponding limitations. ARM2GC [SRH+19] is a framework based on the garbled circuit protocol for secure computation that accepts high-level programs. PCF (Portable Circuit Format) [KSMB13] has introduced a compact representation of Boolean circuits that enables better scaling of secure computation programs. Authors in [KSS12] have shown the evaluation of a circuit with more than a billion gates in the malicious model by parallelizing operations.

Secret Sharing-based Frameworks. The Sharemind framework [BLW08] is based on additive secret sharing over the ring $\mathbb{Z}_{2^{32}}$. The computation is performed with *three* nodes and is secure in the honest-but-curious adversary model where only one node can be corrupted. SEPIA [BSMD10] is a library for privacy-preserving aggregation of data for network security and monitoring. SEPIA is based on Shamir's secret sharing scheme where computation is performed by three (or more) privacy peers. VIFF (Virtual Ideal Functionality Framework) [DGKN09] is a framework that implements asynchronous secure computation protocols and is also based on Shamir's secret sharing. PICCO [ZSB13] is a source-to-source compiler that generates secure multiparty computation protocols from functions written in the `C` language. The output of the compiler is a `C` program that runs the secure computation using linear secret sharing. SPDZ [DPSZ12] is a secure computation protocol based on additive secret sharing that is secure against $n-1$ corrupted computation nodes in the malicious model. Recent work of [AFL+16, FLNW17,ABF+17] introduces an efficient protocol for three-party secure computation. In general, for secret sharing-based frameworks, three (or more) computation nodes need to communicate in the online phase and in some cases, the communication is quadratic in the number of computation nodes. However, in Chameleon, the third node (STP) is not involved in the online phase which

reduces the communication and running time.

While Chameleon offers more flexibility compared to secret-sharing based frameworks, it is also computationally more efficient compared to Sharemind and SEPIA: To perform each multiplication, Sharemind originally[4] needed 6 instances of the Du-Atallah protocol [BLW08] while Chameleon needs 1 (when one operand is shared) or 2 (in the general case where both operands are shared). In SEPIA [BSMD10], all operations are performed modulo a prime number which is less efficient compared to modulo 2^l and also requires multiple multiplications for creating/reconstructing a share.

Mixed Protocol Frameworks. TASTY [HKS+10] is a compiler that can generate mixed protocols based on GC and HE. Several applications have been built that use mixed protocols, e.g., privacy-preserving ridge-regression [NWI+13], matrix factorization [NWI+13], iris and finger-code authentication [BG11], and medical diagnostics [BFK+09].

Recently, a new framework for compiling two-party protocols called EzPC [CGR+17] was presented. EzPC uses ABY as its cryptographic back-end: a simple and easy-to-use imperative programming language is compiled to ABY input. An interesting feature of EzPC is its "cost awareness", i.e. its ability to automatically insert type conversion operations in order to minimize the total cost of the resulting protocol. However, authors claim that ABY's GC engine always provides better performance for binary operations than GMW and thus convert only between A-SS and GC.

Our framework extends the ABY framework [DSZ15]. Specifically, we add support for signed fixed-point numbers which is essential for almost all machine learning applications such as processing deep neural networks. Our framework provides a faster online phase and a more efficient offline phase in terms of computation and communication due to the usage of an STP. Moreover, we implement a highly efficient vector dot product protocol based on correlated

[4]Sharemind replaced the Du-Atallah protocol with a new three-party multiplication protocol [BNTW12]. Due to its symmetry, we cannot modify this protocol to work with only two parties in the online phase as we do for the Du-Atallah protocol in §3.4.2.

randomness generated by an STP.

Automatic Protocol Selection. The authors of [KSS14] propose two methods, one heuristic and one based on integer programming, to find an optimal combination of two secure computation protocols, GC and HE. This methodology has been applied to the ABY framework in CheapSMC [PKUM16]. The current version of Chameleon does not provide automatic protocol selection. However, the methods of [KSS14, PKUM16, CGR+17] can be applied in future work in order to automatically partition Chameleon programs.

Generation of Multiplication Triplets. Very recently, Lu and Sakuma [jLS18] presented an efficient protocol for generating MTs that are specially crafted for matrix multiplications by using additively shared matrices. The protocol results in a significant performance improvement in the offline phase compared to prior work, e.g., up to 110x faster run-time compared to SecureML [MZ17a] and MiniONN [LJLA17a]. However, this protocol is limited to matrix multiplications, whereas Chameleon is generic and thus efficient for any operation.

3.3 The Chameleon Framework

Chameleon comprises of an *offline phase* and an *online phase*. The online phase is a two-party execution model that is run between two parties who wish to perform secure computation on their data. In the offline phase, a Semi-honest Third Party (STP) creates *correlated randomness* together with random seeds and provides it to the two parties as suggested in [Hua12]. We describe how the STP can be implemented in Section 3.3.3 and its role in Section 3.4.2.

The online phase itself consists of three execution environments: GC, GMW, and Additive Secret Sharing (A-SS). We described the functionality of the GC and GMW protocols in Section 2.1 and we detail our implementations of these protocols in Section 3.4.1. We implement two different protocols for the multiplication operation on additive shares: a protocol based on Multiplication Triples (MTs) that we described in Section 2.1.5 and an optimized version of

the Du-Atallah (DA) protocol [DA01] (cf. Section 3.4.2). In Section 3.3.1, we explain how the online phase works. In order to support highly efficient secure computations, all operations that do not depend on the run-time variables are shifted to the offline phase. The only cryptographic operations in the online phase are the Advanced Encryption Standard (AES) operations that are used in GC for which dedicated hardware acceleration is available in many processors via the AES-NI instruction set.

The offline phase includes four tasks: (i) precomputing all required OTs that are used in GC and type conversion protocols, thereby providing a very fast *encryption-free* online phase for OT, (ii) precomputing Arithmetic Multiplication Triples (A-MTs) used in the multiplication of additive secret shares, (iii) precomputing Boolean Multiplication Triples (B-MTs) used in the GMW protocol, and lastly, (iv) precomputing vector dot product shares (VDPS) used in the Du-Atallah protocol [DA01]. In order to reduce the communication in the offline phase from the STP to the two parties, we use the seed expansion technique [DSZ14] for generating A-MTs and B-MTs (cf. Section 3.4.4). We also introduce a novel technique that reduces the communication for generating VDPS (cf. Section 3.4.2).

3.3.1 Chameleon Online Execution Flow

In this section, we provide a high-level description of the execution flow of the online phase. As discussed earlier, linear operations such as ADD, SUB, and MULT are executed in A-SS. The dot product of two vectors of size n is also executed in A-SS which comprises n MULTs and $n-1$ ADDs. Non-linear operations such as CMP, EQ, MUX and bitwise XOR, AND, OR operations are executed in the GMW or GC protocol depending on which one is more efficient. Recall that in order to execute a function using the GMW or GC protocol, the function has to be described as a Boolean circuit.

However, the most efficient Boolean circuit description of a given function is different for the GMW and the GC protocol: In the GC protocol, the computation and communication costs

only depend on the *total number of AND gates* (N_{AND}) in the circuit. Regardless of the number of XOR gates, functionality, and depth of the circuit, GC executes in a *constant* number of rounds. Communication is a linear function of the number of AND gates ($2 \times k \times N_{AND}$). Due to the Half-Gates optimization (cf. Section 3.4.1), computation is bounded by constructing the garbled tables (four fixed-key AES encryptions) and evaluating them (two fixed-key AES encryptions). The GMW protocol, on the other hand, has a different computation and communication model. It needs only bit-level AND and XOR operations for the computation, but one round of communication is needed per layer of AND gates. Therefore, the most efficient representation of a function in the GMW protocol is the one that has minimum *circuit depth*, more precisely, the minimum number of sequentially dependent layers of AND gates. As a result, when the network latency or the depth of the circuit is high, we use GC to execute non-linear functions, otherwise, GMW will be utilized. The computation and communication costs for atomic operations are given in Section 5.6.

The program execution in Chameleon is described as different layers of operations where each layer is most efficiently realized in one of the execution environments. The execution starts from the first layer and the corresponding execution environment. Once all operations in the first layer are finished, Chameleon switches the underlying protocol and continues the process in the second execution environment. Changing the execution environment requires that the type of the shared secrets should be changed in order to enable the second protocol to continue the process. One necessary condition is that the cost of the share type translation must not be very high to avoid diminishing the efficiency achieved by the hybrid execution. For converting between the different sharing types, we use the methods from the ABY framework [DSZ15] which are based on highly efficient OT extensions.

Communication Rounds. The number of rounds that both parties need to communicate in Chameleon depends on the number of switches between execution environments and the depth of the circuits used in the GMW protocol. We want to emphasize that the number of communication rounds does not depend on the size of input data. Therefore, the network latency

added to the execution time is quickly amortized over a high volume of input data.

3.3.2 Security Model

Chameleon is secure against honest-but-curious (HbC), a.k.a. semi-honest, adversaries. This is the standard security model in the literature and considers adversaries that follow the protocol but attempt to extract more information based on the data they receive and process. Honest-but-curious is the security model for the great majority of prior art, e.g., Sharemind [BLW08], ABY [DSZ15], and TinyGarble [SHS+15].

The Semi-honest Third Party (STP) can be either implemented using a physical entity, in a distributed manner using MPC among multiple non-colluding parties, using trusted hardware (hardware security modules or smartcards [DSZ14]), or using trusted execution environments such as Intel SGX [BBB+17]. In case the STP is implemented as a separate physical computation node, our framework is secure against semi-honest adversaries with an honest majority. The latter is identical to the security model considered in Sharemind [BLW08]. In Section 3.2, we list further works based on similar assumptions. Please note that we introduce a new and more practical *computational* model that is superior to Sharemind since only two primary parties are involved in the online execution. This results in a significantly faster run-time while better matching real-world requirements.

3.3.3 Semi-honest Third Party (STP)

In Chameleon, the STP is only involved in the offline phase in order to generate correlated randomness [Hua12]. It is not involved in the online phase and thus does not receive any information about the two parties' inputs nor the program being executed. The only exception is when computing VDPS for the Du-Atallah protocol: the STP needs to know the size of the vectors in each dot product beforehand. Since the security model in Chameleon is HbC with

honest majority, some information can be revealed if the STP colludes with either party.

In order to prevent the STP from observing communication between the two parties, authenticated encryption is added to the communication channel. Also the communication between the STP and the two parties is encrypted, so they cannot reconstruct the other party's private inputs from observed messages.

3.4 Chameleon Design and Implementation

In this section, we provide a detailed description of the different components of Chameleon. Chameleon is written in `C++` and accepts the program written in `C++`. The implementation of the GC and GMW engines is covered in Section 3.4.1 and the A-SS engine is described in Section 3.4.2. Section 3.4.3 illustrates how Chameleon supports signed fixed-point representation. The majority of cryptographic operations is shifted from the online to the offline phase. Thus, in Section 3.4.4, we describe the process of generating Arithmetic/Boolean Multiplication Triples (A-MTs/B-MTs). Section 3.4.5 provides our STP-based implementation for fast Oblivious Transfer and finally the security justification of Chameleon is given in Section 3.4.6.

3.4.1 GC and GMW Engines

Chameleon's implementation of the GC and GMW protocol is based on ABY [DSZ15]. Therefore, the input to the engines is the topologically sorted list of Boolean gates in the circuit as an `.aby` file. The GC engine includes the most recent optimizations: Free-XOR [KS08b], fixed-key AES garbling [BHKR13], and Half-Gates [ZRE15]. We synthesized GC-optimized circuits for many primitive functions. Likewise, for the GMW engine all circuits are depth-optimized as described in [DDK$^+$15] to incur the least latency during the protocol execution. A user can simply use these circuits by calling regular functions in `C++`.

3.4.2 A-SS Engine

In Chameleon, linear operations, i.e., ADD, SUB, MULT, are performed using additive secret sharing in the ring $\mathbb{Z}_{2^l}$. We discussed in Section 2.1.5 how to perform a single MULT using a multiplication triple. However, there are other methods to perform a MULT: (i) The protocol of [BELO16] has very low communication in the online phase. However, in contrast to our computation model, it requires STP interaction with the other two parties in the online phase. (ii) The Du-Atallah protocol [DA01] is another method to perform multiplication on additive shared values which we describe next.

The Du-Atallah Multiplication Protocol [DA01]. In this protocol, two parties $\mathcal{P}_0$ (holding x) and $\mathcal{P}_1$ (holding y) together with a third party $\mathcal{P}_2$ can perform the multiplication $z = x \times y$. At the end of this protocol, z is additively shared between all *three* parties. The protocol works as follows:

1. $\mathcal{P}_2$ randomly generates $a_0, a_1 \in_R \mathbb{Z}_{2^l}$ and sends a_0 to $\mathcal{P}_0$ and a_1 to $\mathcal{P}_1$.

2. $\mathcal{P}_0$ computes $(x + a_0)$ and sends it to $\mathcal{P}_1$. Similarly, $\mathcal{P}_1$ computes $(y + a_1)$ and sends it to $\mathcal{P}_0$.

3. $\mathcal{P}_0$, $\mathcal{P}_1$, and $\mathcal{P}_2$ can compute their share as $\langle z \rangle_0^A = -a_0 \times (y + a_1)$, $\langle z \rangle_1^A = y \times (x + a_0)$, and $\langle z \rangle_2^A = a_0 \times a_1$, respectively.

It can be observed that the results are true additive shares of z: $\langle z \rangle_0^A + \langle z \rangle_1^A + \langle z \rangle_2^A = z$. Please note that this protocol computes shares of a multiplication of two numbers held by two parties in *cleartext*. In the general case, where both x and y are additively shared between two parties ($\mathcal{P}_0$ holds $\langle x \rangle_0^A$, $\langle y \rangle_0^A$ and $\mathcal{P}_1$ holds $\langle x \rangle_1^A$, $\langle y \rangle_1^A$), the multiplication can be computed as $z = x \times y = (\langle x \rangle_0^A + \langle x \rangle_1^A) \times (\langle y \rangle_0^A + \langle y \rangle_1^A)$. The two terms $\langle x \rangle_0^A \times \langle y \rangle_0^A$ and $\langle x \rangle_1^A \times \langle y \rangle_1^A$ can be computed locally by $\mathcal{P}_0$ and $\mathcal{P}_1$, respectively. *Two* instances of the Du-Atallah protocol are needed to compute shares of $\langle x \rangle_0^A \times \langle y \rangle_1^A$ and $\langle x \rangle_1^A \times \langle y \rangle_0^A$. Please note that $\mathcal{P}_i$ should not learn $\langle x \rangle_{1-i}^A$ and $\langle y \rangle_{1-i}^A$,

otherwise, secret values x and/or y are revealed to $\mathcal{P}_i$. At the end, $\mathcal{P}_0$ has

$$\langle x\rangle_0^A \times \langle y\rangle_0^A,\ \left\langle \langle x\rangle_0^A \times \langle y\rangle_1^A \right\rangle_0^A,\ \left\langle \langle x\rangle_1^A \times \langle y\rangle_0^A \right\rangle_0^A$$

and $\mathcal{P}_1$ has

$$\langle x\rangle_1^A \times \langle y\rangle_1^A,\ \left\langle \langle x\rangle_0^A \times \langle y\rangle_1^A \right\rangle_1^A,\ \left\langle \langle x\rangle_1^A \times \langle y\rangle_0^A \right\rangle_1^A,$$

where $\langle z\rangle_0^A$, $\langle z\rangle_1^A$ are the summations of each party's shares.

The Du-Atallah protocol was used in Sharemind [BLW08] where there are three active computing nodes that are involved in the online phase, whereas, in Chameleon, the third party (STP) is only involved in the offline phase. This problem can be solved since the role of $\mathcal{P}_2$ can be shifted to the offline phase as follows: (i) Step one of the Du-Atallah protocol can be computed in the offline phase for as many multiplications as needed. (ii) In addition, $\mathcal{P}_2$ randomly generates another l-bit number a_2 and computes $a_3 = (a_0 \times a_1) - a_2$. $\mathcal{P}_2$ sends a_2 to $\mathcal{P}_0$ and a_3 to $\mathcal{P}_1$ in the offline phase. During the online phase, both parties additionally add their new shares (a_2 and a_3) to their shared results: $\langle z\rangle_{0,new}^A = \langle z\rangle_0^A + a_2$ and $\langle z\rangle_{1,new}^A = \langle z\rangle_1^A + a_3$.

Security. This modification is perfectly secure since $\mathcal{P}_0$ has received a true random number and $\mathcal{P}_1$ has received a_3 which is an additive share of $(a_0 \times a_1)$. Since a_2 has uniform distribution, the probability distribution of a_3 is also uniform [BLW08] and as a result, $\mathcal{P}_1$ cannot infer additional information.

Du-Atallah Protocol with one cleartext operand. In many cases, the computation model is such that one operand x is held in cleartext by one party, e.g., $\mathcal{P}_0$, and the other operand y is shared among two parties: $\mathcal{P}_0$ has $\langle y\rangle_0^A$ and $\mathcal{P}_1$ has $\langle y\rangle_1^A$. This situation repeatedly arises when the intermediate result is multiplied by one of the party's inputs which is not shared. In this case, only one instance of the Du-Atallah protocol is needed to compute $x \times \langle y\rangle_1^A$. As analyzed in this section, employing this variant of the Du-Atallah protocol is more efficient than the protocol based on MTs. Please note that in order to utilize MTs, both operands need to be shared among

the two parties first, which, as we argue here, is inefficient and unnecessary. Table 3.1 summarizes the computation and communication costs for the Du-Atallah protocol and the protocol based on MTs (Section 2.1.5). As can be seen, online computation and communication is improved by factor 2x. Also, the offline communication is improved by factor 3x. Unfortunately, using the Du-Atallah protocol in this format will reduce the efficiency of vector dot product computation in Chameleon. Please note that it is no longer possible to perform a complete dot product of two vectors by two parties only. The reason is that the third share ($\langle z \rangle_2^A = a_0 \times a_1$) is shared between two parties ($\mathcal{P}_0$ and $\mathcal{P}_1$). However, this problem can be solved by a modification which we describe next.

Table 3.1: Summary of properties of the Du-Atallah multiplication protocol and the protocol based on Multiplication Triples.

Protocol	**# MULT ops**	**Online Comm.**	**Offline Comm.**	**Rounds**
Multiplication Triple	(3,4)	$2 \cdot l$	$3 \cdot l$	2^*
Du-Atallah	(1,2)	l	$2 \cdot l$	1

Du-Atallah Protocol and Vector Dot Product. We further modify the optimized Du-Atallah protocol such that the complete vector dot product is efficiently processed. The idea is that instead of the STP additively sharing its shares, it first sums its shares and then sends the additively shared versions to the two parties. Consider vectors of size n. The STP needs to generate n different a_0 and a_1 as a list for a single vector multiplication. We denote the j^{th} member of the list as $[a_0]_j$ and $[a_1]_j$. Our modification requires that the STP generates a single l-bit value a_2 and sends it to $\mathcal{P}_0$. The STP also computes $a_3 = \sum_{i=0}^{n-1} [a_0]_j \times [a_1]_j - a_2$ and sends it to $\mathcal{P}_1$. We call a_2 and a_3 the *Vector Dot Product Shares* (VDPS). This requires that the STP knows the size of the array in the offline phase. Since the functionality of the computation is not secret, we can calculate the size and number of all dot products in the offline phase and ask for the corresponding random shares from the STP.

Reducing Communication. A straightforward implementation of the offline phase of

the Du-Atallah protocol requires that the STP sends $\sim n$ random numbers of size l ($[a_0]_j$ and $[a_1]_j$) to $\mathcal{P}_0$ and $\mathcal{P}_1$ for a single dot product of vectors of size n. However, we suggest reducing the communication using a Pseudo Random Generator (PRG) for generating the random numbers as was proposed in [DSZ14]. Instead of sending the complete list of numbers to each party, the STP can create and send random PRG seeds for each string to the parties such that each party can create $[a_0]_j$ and $[a_1]_j$ locally using the PRG. For this purpose, we implement the PRG using Advanced Encryption Standard (AES), a low-cost block cipher, in counter mode (AES CTR-DRBG). Our implementation follows the description of the NIST Recommendation for DRBGs [BK15a]. From a 256-bit seed, AES CTR-DRBG can generate 2^{63} indistinguishable random bits. If more than 2^{63} bits are needed, the STP sends more seeds to the parties. The STP uses the same seeds in order to generate a_2 and a_3 for each dot product. Therefore, the communication is reduced from $n \times l$ bits to sending a one-time 256-bit seed and an l-bit number per single dot product.

Performance Evaluation. We give an empirical performance evaluation of our optimized VDP protocol in Section 3.5: the evaluated SVM classification mainly consists of a VDP computation together with a negligible subtraction and comparison operation.

3.4.3 Supporting Signed Fixed-point Numbers

Chameleon supports Signed Fixed-point Numbers (SFN) in addition to integer operations. Supporting SFN requires not only that all three secure computation protocols (GC, GMW, and Additive SS) support SFN but also the secret translation protocols to be compatible. We note that the current version of the ABY framework only supports unsigned integers and IEEE 754 floating point numbers [DDK$^+$15]. We added an abstraction layer to the ABY framework such that it supports SFN.

All additive secret sharing protocols only support unsigned integer values. However, in this section, we describe how such protocols can be modified to support *signed fixed-point*

numbers. Supporting *signed integers* can be done by representing numbers in two's complement format. Consider the ring $\mathbb{Z}_{2^l}$ which consists of unsigned integer numbers $\{0,1,2,...,2^{l-1}-1,2^{l-1},...,2^l-1\}$. We can perform signed operations by simply *interpreting* these numbers as the two's complement format: $\{0,1,2,...,2^{l-1}-1,-2^{l-1},...,-1\}$. By doing so, signed operations work seamlessly.

In order to support fixed-point precision, one solution is to interpret signed integers as signed fixed-point numbers. Each number is represented in two's complement format with the Most Significant Bit (MSB) being the sign bit. There are α and β bits for integer and fraction parts, respectively. Therefore, the total number of bits is equal to $\gamma = 1+\alpha+\beta$. While this works perfectly for addition and subtraction, it cannot be used for multiplication. The reason is that when multiplying two numbers in a ring, the rightmost $2\times\beta$ bits of the result correspond to the fraction part while β bits of the MSBs are overflown and discarded. Our solution to this problem is to perform all operations in the ring $\mathbb{Z}_{2^l}$ where $l = \gamma+\beta$. After each multiplication, we shift the result β bits to the right while replicating the sign bit for β MSBs. While bitshifting by a constant and sign bit replication is essentially free in GC/GMW, it is non-trivial in additive sharing. Thus, a conversion from additive sharing to GC/GMW and back is required between multiplications. Compared to [MZ17a], where the authors apply a similar approach to fixed-point arithmetic but simply truncate additive shares, this prevents introducing up to 1 bit inaccuracy per multiplication. The overhead for the conversions is smaller than adapting the approach of [RRK18], where the authors naively apply the same method as used for floating-point arithmetic in ABY [DDK+15], i.e., they use hardware compilers to generate circuits which perform fixed-point arithmetic in GC. Following the observation that in GC the overhead for multiplication even for integer numbers is large, we expect our mixed-protocol approach to greatly outperform their implementation. Please note that for the machine learning applications no preventable overhead for protocol conversion occurs: between all multiplications a non-linear function is computed, which requires conversion to GC/GMW anyhow.

This assumes that in addition to the support by the computation engines, share translation protocols actually work correctly. Share translation from GC to GMW works fine as it operates on bit-level and is transparent to the number representation format. Share translation from GC/GMW to additive sharing either happens using a subtraction circuit or OT. In the first case, the result is valid since the subtraction of two signed fixed-point numbers in two's complement format is identical to subtracting two unsigned integers. In the second case, OT is on bit-level and again transparent to the representation format. In Chameleon, as in ABY, we use the OT method for share translation from GC/GMW to additive due to reduced complexity. Finally, share translation from additive sharing to GC/GMW is correct because it uses an addition circuit, which is identical for unsigned integers and signed fixed-point numbers.

Floating Point Operations. Chameleon supports floating point operations by performing all computations in the GC or GMW protocol as described in [DDK+15] for ABY. A future direction of this work can be to break down the primitive floating point operations, e.g., ADD, MULT, SUB, etc. into smaller atomic operations based on integer values. Consequently, one can perform the linear operations in the ring and non-linear operations in GC/GMW, providing a faster execution for floating-point operations.

Most methods for secure computation on floating and fixed point numbers proposed in the literature were realized in Shamir's secret sharing scheme, e.g. [CS10, ABZS13, ZSB13, KW14, PS15], but some of them also in GC [PS15], GMW [DDK+15], and HE [LDD+16] based schemes. The quality of the algorithms varies from self-made to properly implemented IEEE 754 algorithms, such as in [PS15, DDK+15]. The corresponding software implementations were done either in the frameworks Sharemind [BLW08] and PICCO [ZSB13], or as standalone applications. For fixed-point arithmetics, Aliasgari et al. [ABZS13] proposed algorithms that outperform even integer arithmetic for certain operations. As a future direction of this work, we plan to integrate their methodology in Chameleon.

3.4.4 Generating Multiplication Triples

As we discussed in Section 2.1.5, each multiplication on additive secret shares requires an Arithmetic Multiplication Triple (A-MT) and one round of communication. Similarly, evaluating each AND gate in the GMW protocol requires a Boolean Multiplication Triple (B-MT) [DSZ14]. In the offline phase, we calculate the number of MTs ($N_{\text{A-MT}}$ and $N_{\text{B-MT}}$). The STP precomputes all MTs needed and sends them to both parties. More precisely, to generate A-MTs, the STP uses a PRG to produce five l-bit random numbers corresponding to a_0, b_0, c_0, a_1, and b_1. We denote the j^{th} triple with $[.]_j$. Therefore, the STP completes MTs by computing c_1's as $[c_1]_j = ([a_0]_j + [a_1]_j) \times ([b_0]_j + [b_1]_j) - [c_0]_j$. Finally, the STP sends $[a_0]_j, [b_0]_j$, and $[c_0]_j$ to the first party and $[a_1]_j, [b_1]_j$, and $[c_1]_j$ to the second party for $j = 1, 2, ..., N_{\text{A-MT}}$. Computing B-MTs is also very similar with the only differences that all numbers are 1-bit and $[c_1]_j$ is calculated as $[c_1]_j = ([a_0]_j \oplus [a_1]_j) \wedge ([b_0]_j \oplus [b_1]_j) \oplus [c_0]_j$.

Reducing Communication. A basic implementation of precomputing A-MTs and B-MTs requires communication of $3 \times l \times N_{\text{A-MT}}$ and $3 \times N_{\text{B-MT}}$ bits from the STP to each party, respectively. However, similar to the idea of [DSZ14] presented in Section 3.4.2, we use a PRG to generate random strings from seeds locally for each party. To summarize the steps: the STP

1. generates two random seeds: seed_0 for generating $[a_0]_j, [b_0]_j$, and $[c_0]_j$ and seed_1 for $[a_1]_j$ and $[b_1]_j$;

2. computes $[c_1]_j = ([a_0]_j + [a_1]_j) \times ([b_0]_j + [b_1]_j) - [c_0]_j$ for $j = 1, 2, ..., N_{A-MT}$;

3. sends seed_0 to the first party and seed_1 together with the list of $[c_1]_j$ to the second party.

After receiving the seeds, both parties locally generate their share of the triples using the same PRG. This method reduces the communication from $3 \times l \times N_{\text{A-MT}}$ to 256 and $256 + l \times N_{\text{A-MT}}$ bits for the first and second party, respectively. The STP follows a similar process to generate B-MTs. Figure 3.1 illustrates the seed expansion idea to generate MTs [DSZ14].

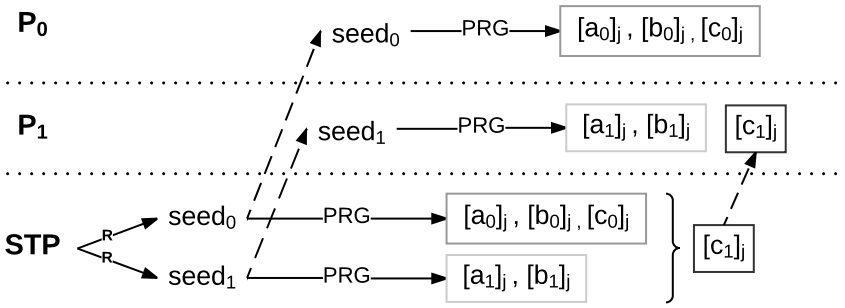


Figure 3.1: Seed expansion process to precompute A-MTs/B-MTs with low communication.

3.4.5 Fast STP-aided Oblivious Transfer

Utilizing the idea of correlated randomness [Hua12], we present an efficient and fast protocol for Oblivious Transfer that is aided by the Semi-honest Third Party (STP). Our protocol comprises an offline phase (performed by the STP) and an online phase (performed by the two parties). The protocol is described for one 1-out-of-2 OT. The process repeats for as many OTs as required. In the offline phase, the STP generates random masks q_0, q_1 and a random bit r and sends q_0, q_1 to the sender and r, q_r to the receiver. In the online phase, two parties execute the online phase of Beaver's OT precomputation protocol [Bea95] described in Figure 3.2. Please note that all OTs in Chameleon including OTs used in GC and secret translation from GC/GMW to Additive are implemented as described above.

Sender		Receiver
Has: messages q_0, q_1 Input: messages m_0, m_1 Output: -		Has: message q_r and r Input: choice bit b Output: m_b
	$\xleftarrow{b'}$	$b' = r \oplus b$
$(s_0, s_1) =$ $\begin{cases} (q_0 \oplus m_0, q_1 \oplus m_1) \text{ if } b' = 0 \\ (q_0 \oplus m_1, q_1 \oplus m_0) \text{ if } b' = 1 \end{cases}$	$\xrightarrow{(s_0, s_1)}$	$m_b = s_r \oplus q_r$

Figure 3.2: Beaver's OT precomputation protocol [Bea95].

Reducing Communication. Similar to the idea discussed in Section 3.4.4, the STP does not actually need to send the list of (q_0, q_1) to the sender and r to the receiver. Instead, it generates two random seeds and sends them to the two parties. The STP only needs to send the full list of

q_r to the receiver.

3.4.6 Security

Chameleon is based on the ABY framework [DSZ15] where we replace the interactive offline phase with the following STP-based protocols: (i) STP-aided OTs (cf. Section 3.4.5) are implemented via Beaver's OT precomputation [Bea95] where the original OTs are sent by the STP, which is trivially secure. (ii) STP-aided generation of MTs (cf. Section 3.4.4) was proven secure in [Hua12, DSZ14]. (iii) STP-aided multiplication (cf. Section 3.4.2) is done based on an STP-aided extension of the Du-Atallah multiplication protocol [DA01] for which we have argued security already in Section 3.4.2; all further optimizations are simply a compression of the data sent by the STP and hence do not leak any additional information. In summary, security of Chameleon follows from the security of ABY and the security of our STP-based protocols, so we can state the following theorem.

Theorem 1 *Chameleon's STP-based protocols are secure against HbC adversaries under the assumption that at most one of the two parties is passively corrupted and none of them colludes with the STP.*

3.5 Experimental Results

We experimentally verify the efficiency of the Chameleon by securely evaluating several machine learning models such as support vector machines and deep and convolutional neural network. Our experimental setup is performed for 128-bit security parameter on machines equipped with Intel Core i7-4790 CPUs @ 3.6GHz and 16GB of RAM with AES-NI support. In this section, a highlight of the experimental results is provided and performance of Chameleon is compared with the prior art.

The first benchmark is a convolutional neural network with one convolution layer and two fully connected layer which classifies hand-written digits of the MNIST dataset. Our proposed framework is able to classify an image in 2.24 seconds with 10.5 MB of communication. The classification accuracy is 99%.

The second benchmark is a deeper CNN with seven convolution layers, two MaxPooling layers, and one fully connected layer. The CNN model is able to classify images from CIFAR-10 dataset. A single secure inference takes 52.67 seconds and requires 2.65 GB of communication.

The third benchmark is a support vector machine model. Classifying an input with 10 features, takes 9.88 ms and 6.5 KB of communication. Extending the feature size to 1000, the runtime increases to 11.42 ms and the communication increases to 30.3 KB. More details about the experimental setup and evaluation benchmarks are discussed in [RWT+18a].

3.6 Summary

We introduced Chameleon, a novel hybrid (mixed-protocol) secure computation framework based on ABY [DSZ15] that achieves unprecedented performance by (i) providing an optimized vector dot product protocol for fast matrix multiplications and (ii) employing a semi-honest third party in the offline phase for generating correlated randomness that is used for pre-computing OTs and multiplication triples. In contrast to previous state-of-the-art frameworks, Chameleon supports signed fixed-point numbers. We evaluated our framework on convolutional neural networks where it can process an image of hand-written digits 133x faster than the prior art Microsoft CryptoNets [DGBL+16a] and 4.2x faster than the most recent MiniONN [LJLA17a].

3.7 Acknowledgements

This chapter, in part, has been published at the Proceedings of 2018 ACM ASIA Conference on Computer and Communications Security (AsiaCCS) and appeared as: M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, Farinaz Koushanfar, "Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications". The dissertation author was the primary author of this material. Christian Weinert and Oleksandr Tkachenko are the main contributor of the implementation and experimental results of this work.

Chapter 4

XONN: Efficient Neural Network Transformation for Oblivious Inference

Advancements in deep learning enable cloud servers to provide inference-as-a-service for clients. In this scenario, clients send their raw data to the server to run the deep learning model and send back the results. One standing challenge in this setting is to ensure the privacy of the clients' sensitive data. Oblivious inference is the task of running the neural network on the client's input without disclosing the input or the result to the server. This chapter introduces XONN (pronounced /zʌn/), a novel end-to-end framework based on Yao's Garbled Circuits (GC) protocol, that provides a paradigm shift in the conceptual and practical realization of oblivious inference. In XONN, the costly matrix-multiplication operations of the deep learning model are replaced with `XNOR` operations that are essentially free in GC. We further provide a novel algorithm that customizes the neural network such that the runtime of the GC protocol is minimized without sacrificing the inference accuracy.

We design a user-friendly high-level API for XONN, allowing expression of the deep learning model architecture in an unprecedented level of abstraction. We further provide a compiler to translate the model description from high-level Python (i.e., Keras) to that of XONN.

Extensive proof-of-concept evaluation on various neural network architectures demonstrates that XONN outperforms prior art such as Gazelle (USENIX Security'18) by up to 7×, MiniONN (ACM CCS'17) by 93×, and SecureML (IEEE S&P'17) by 37×. State-of-the-art frameworks require one round of interaction between the client and the server for each layer of the neural network, whereas, XONN requires a *constant* round of interactions for *any* number of layers in the model. XONN is first to perform oblivious inference on Fitnet architectures with up to 21 layers, suggesting a new level of scalability compared with state-of-the-art. Moreover, we evaluate XONN on four datasets to perform privacy-preserving medical diagnosis. The datasets include breast cancer, diabetes, liver disease, and Malaria.

4.1 Introduction

Oblivious inference, i.e., privacy-preserving inference, enables the execution of an AI model on an encrypted query. Oblivious inference has many applications across different industries. Consider a DL model used in a medical task in which a health service provider withholds the prediction model. Patients submit their plaintext medical information to the server, which then uses the sensitive data to provide a medical diagnosis based on inference obtained from its proprietary model. A naive solution to ensure patient privacy is to allow the patients to receive the DL model and run it on their own trusted platform. However, this solution is not practical in real-world scenarios because: (i) The DL model is considered an essential component of the service provider's intellectual property (IP). Companies invest a significant amount of resources and funding to gather the massive datasets and train the DL models; hence, it is important to service providers not to reveal the DL model to ensure their profitability and competitive advantage. (ii) The DL model is known to reveal information about the underlying data used for training [TZJ+16]. In the case of medical data, this reveals sensitive information about other patients, violating HIPAA and similar patient health privacy regulations.

Several solutions for oblivious inference have been proposed that utilize one or more cryptographic tools such as Homomorphic Encryption (HE) [BV14, BGV14], Garbled Circuits (GC) [Yao86b], Goldreich-Micali-Wigderson (GMW) protocol [GMW87], and Secret Sharing (SS). Each of these cryptographic tools offer their own characteristics and trade-offs. For example, one major drawback of HE is its *computational complexity*. HE has two main variants: Fully Homomorphic Encryption (FHE) [BV14] and Partially Homomorphic Encryption (PHE) [BGV14, Pai99]. FHE allows computation on encrypted data but is computationally very expensive. PHE has less overhead but only supports a subset of functions or depth-bounded arithmetic circuits. The computational complexity drastically increases with the circuit's depth. Moreover, non-polynomial functionalities such as the ReLU activation function in DL cannot be supported.

GC, on the other hand, can support an arbitrary functionality while requiring only a *constant* round of interactions regardless of the depth of the computation. However, it has a high communication cost and a significant overhead for multiplication. More precisely, performing multiplication in GC has quadratic computation and communication complexity with respect to the bit-length of the input operands. It is well-known that the complexity of the contemporary DL methodologies is dominated by matrix-vector multiplications. GMW needs less communication than GC but requires many rounds of *interactions* between the two parties.

A standalone SS-based scheme provides a computationally inexpensive multiplication yet requires three or more independent (non-colluding) computing servers, which is a strong assumption. Mixed-protocol solutions have been proposed with the aim of utilizing the best characteristics of each of these protocols [RWT$^+$18a, MZ17b, LJLA17b, JVC18]. They require secure conversion of secrets from one protocol to another in the middle of execution. Nevertheless, it has been shown that the cost of secret conversion is paid off in these hybrid solutions. Roughly speaking, the number of interactions between server and client (i.e., round complexity) in existing hybrid solutions is *linear* with respect to the depth of the DL model. Since depth is a major contributor to the deep learning accuracy [SLJ$^+$15], scalability of the mixed-protocol solutions

with respect to the number of layers remains an unsolved issue for more complex, many-layer networks.

This chapter introduces XONN, a novel end-to-end framework which provides a paradigm shift in the conceptual and practical realization of privacy-preserving interference on deep neural networks. The existing work has largely focused on the development of customized security protocols while using conventional fixed-point deep learning algorithms. XONN, for the first time, suggests leveraging the concept of the Binary Neural Networks (BNNs) in conjunction with the GC protocol. In BNNs, the weights and activations are restricted to binary (i.e, ± 1) values, substituting the costly multiplications with simple `XNOR` operations during the inference phase. The `XNOR` operation is known to be *free* in the GC protocol [KS08a]; therefore, performing oblivious inference on BNNs using GC results in the removal of costly multiplications. Using our approach, we show that oblivious inference on the standard DL benchmarks can be performed with minimal, if any, decrease in the prediction accuracy.

We emphasize that an effective solution for oblivious inference should take into account the deep learning algorithms and optimization methods that can tailor the DL model for the security protocol. Current DL models are designed to run on CPU/GPU platforms where many multiplications can be performed with high throughput, whereas, bit-level operations are very inefficient. In the GC protocol, however, bit-level operations are inexpensive, but multiplications are rather costly. As such, we propose to train deep neural networks that involve many bit-level operations but *no* multiplications in the inference phase; using the idea of learning binary networks, we achieve an average of $21\times$ reduction in the number of gates for the GC protocol.

We perform extensive evaluations on different datasets. Compared to the Gazelle [JVC18] (the prior best solution) and MiniONN [LJLA17b] frameworks, we achieve $7\times$ and $93\times$ lower inference latency, respectively. XONN outperforms DeepSecure [RRK18] (prior best GC-based framework) by $60\times$ and CryptoNets [DGBL$^+$16b], an HE-based framework, by $1859\times$. Moreover, our solution renders a *constant* round of interactions between the client and the server,

which has a significant effect on the performance on oblivious inference in Internet settings. We highlight our contributions as follows:

- Introduction of XONN, the *first* framework for privacy preserving DNN inference with a *constant* round complexity that does not need expensive matrix multiplications. Our solution is the first that can be scalably adapted to ensure security against malicious adversaries.

- Proposing a novel conditional addition protocol based on Oblivious Transfer (OT) [Rab05], which optimizes the costly computations for the network's input layer. Our protocol is 6× faster than GC and can be of independent interest. We also devise a novel network trimming algorithm to remove neurons from DNNs that minimally contribute to the inference accuracy, further reducing the GC complexity.

- Designing a high-level API to readily automate fast adaptation of XONN, such that users only input a high-level description of the neural network. We further facilitate the usage of our framework by designing a compiler that translates the network description from Keras to XONN.

- Proof-of-concept implementation of XONN and evaluation on various standard deep learning benchmarks. To demonstrate the scalability of XONN, we perform oblivious inference on neural networks with as many as 21 layers for the first time in the oblivious inference literature.

4.2 Related Work

One of the early solutions for oblivious inference is proposed by Barni et al. [BOP06] that leverages homomorphic encryption. Howerver, this work leaks some information about the intermediate states of the computation. Orlandi et al. [OPB07] improve upon this work and avoid

any information leakage. In [SS08], authors propose a mechanism to also keep the topology of the neural network secret, at the cost of more computation overhead. A solution based on additively homomorphic encryption and garbled circuits is presented in [BFL+11].

CryptoNets [DGBL+16b] suggests the adaptation of Leveled Homomorphic Encryption (LHE) to perform oblivious inference. LHE is a variant of Partially HE that enables evaluation of depth-bounded arithmetic circuits. DeepSecure [RRK18] is a privacy-preserving DL framework that relies on the GC protocol. CryptoDL [HTGW18] improves upon CryptoNets [DGBL+16b] and proposes more efficient approximation of the non-linear functions using low-degree polynomials. Their solution is based on LHE and uses mean-pooling in replacement of the max-pooling layer. Chou et al. propose to utilize the sparsity within the DL model to accelerate the inference [CBL+18].

SecureML [MZ17b] is a privacy-preserving machine learning framework based on homomorphic encryption, GC, and secret sharing. SecureML also uses customized activation functions and supports privacy-preserving training in addition to inference. Two non-colluding servers are used to train the DL model where each client XOR-shares her input and sends the shares to both servers. MiniONN [LJLA17b] is a mixed-protocol framework for oblivious inference. The underlying cryptographic protocols are HE, GC, and secret sharing.

Chameleon [RWT+18a] is a more recent mixed-protocol framework for machine learning, i.e., Support Vector Machines (SVMs) as well as DNNs. Authors propose to perform low-depth non-linear functions using the Goldreich-Micali-Wigderson (GMW) protocol [GMW87], high-depth functions by the GC protocol, and linear operations using additive secret sharing. Moreover, they propose to use correlated randomness to more efficiently compute linear operations. EzPC [CGR+17] is a secure computation framework that enables users to write high-level programs and create a protocol based on both Boolean and Arithmetic shares. The back-end cryptographic engine is based on the ABY framework [DSZ15]. HyCC [BDK+18] is a high-level secure computation framework that complies ANSI C programs into efficient hybrid protocols.

Shokri and Shmatikov [SS15] proposed a solution for privacy-preserving collaborative deep learning where the training data is distributed among many parties. Their approach, enables clients to train their local model on their own training data and update the central model's parameters held by a central server. However, it has been shown that a malicious client can learn significant information about the other client's private data [HAPC17]. In [IKR+19], a framework for secure collaborative learning is proposed which is based on the BMR protocol and high-dimensional (HD) computing. Google [BIK+17] has recently introduced a new approach for securely aggregating the parameter updates from multiple users. However, none of these approaches [SS15, BIK+17] study the oblivious inference problem. An overview of related frameworks is provided in [RRK19, RK18].

Frameworks such as ABY^3 [MR18] and SecureNN [WGC18] have different computation models and they rely on three (or four) parties during the oblivious inference. In contrast, XONN does not require an additional server for the computation. In E2DM framework [JKLS18a], the model owner can encrypt and outsource the model to an untrusted server to perform oblivious inference. Concurrently and independently of ours, in TAPAS [SKGK18], Sanyal et al. study the binarization of neural networks in the context of oblivious inference. They report inference latency of 147 seconds on MNIST dataset with 98.6% prediction accuracy using custom CNN architecture. However, as we show in Section 4.7 (BM3 benchmark), XONN outperforms TAPAS by close to *three orders of magnitude*.

Gazelle [JVC18] is the previously most efficient oblivious inference framework. It is a mixed-protocol approach based on additive HE and GC. In Gazelle, convolution operations are performed using the packing property of HE. In this approach, many numbers are packed inside a single ciphertext for faster convolutions. In Section 4.3, we briefly discuss one of the essential requirements that the Gazelle protocol has to satisfy in order to be secure, namely, *circuit privacy*.

High-Level Comparison. In contrast to prior art, we propose a DL-secure computation co-design approach. To the best of our knowledge, DeepSecure [RRK18] is the only solution

that preprocesses the data and network before the secure computation protocol. However, the preprocessing step reveals some information about the network parameters and structure of data. Moreover, another advantage of XONN is the constant round complexity regardless of the number of layers in the NN. It has been shown that round complexity is one of the important criteria in designing secure computation protocols [BELO16] since the performance can significantly be reduced where the network latency is high. As we show in Section 4.7, XONN outperforms all previous solutions in inference latency. Table 4.1 summarizes a high-level comparison between state-of-the-art oblivious inference frameworks.

Table 4.1: High-Level Comparison of oblivious inference frameworks. "**C**"onstant round complexity. "**D**"eep learning/secure computation co-design. "**I**"ndependence of secondary server. "**U**"pgradeable to malicious security using standard solutions. "**S**"upporting any non-linear layer.

Framework	**Crypto. Protocol**	**C**	**D**	**I**	**U**	**S**
CryptoNets [DGBL$^+$16b]	HE	✓	✗	✓	✗	✗
DeepSecure [RRK18]	GC	✓	✓	✓	✓	✓
SecureML [MZ17b]	HE, GC, SS	✗	✗	✗	✗	✗
MiniONN [LJLA17b]	HE, GC, SS	✗	✗	✓	✗	✓
Chameleon [RWT$^+$18a]	GC, GMW, SS	✗	✗	✗	✗	✓
EzPC [CGR$^+$17]	GC, SS	✗	✗	✓	✗	✓
Gazelle [JVC18]	HE, GC, SS	✗	✗	✓	✗	✓
XONN (This work)	GC, SS	✓	✓	✓	✓	✓

4.3 Circuit Privacy

In Gazelle [JVC18], for each linear layer, the protocol starts with a vector $\mathbf{m}$ that is secret-shared between client $\mathbf{m_1}$ and server $\mathbf{m_2}$ ($\mathbf{m} = \mathbf{m_1} + \mathbf{m_2}$). The protocol outputs the secret shares of the vector $\mathbf{m}' = A \cdot \mathbf{m}$ where A is a matrix known to the server but not to the client. The protocol has the following procedure: (i) Client generates a pair (pk, sk) of public and secret keys of an additive homomorphic encryption scheme HE. (ii) Client sends $\mathsf{HE}.Enc_{pk}(\mathbf{m_1})$ to the server. Server adds its share ($\mathbf{m_2}$) to the ciphertext and recovers encryption of $\mathbf{m}$: $\mathsf{HE}.Enc_{pk}(\mathbf{m})$. (iii) Server homomorphically evaluates the multiplication with A and obtains the encryption

of $\mathbf{m}'$. (iv) Server secret shares $\mathbf{m}'$ by sampling a random vector $\mathbf{r}$ and returns ciphertext $\mathbf{c} = \mathsf{HE}.Enc_{pk}(\mathbf{m}' - \mathbf{r})$ to the client. The client can decrypt $\mathbf{c}$ using private key *sk* and obtain $\mathbf{m}' - \mathbf{r}$.

Gazelle uses the Brakerski-Fan-Vercauteren (BFV) scheme [Bra12, FV12a]. However, the vanilla BFV scheme does not provide circuit privacy. At high-level, the circuit privacy requirement states that the ciphertext $\mathbf{c}$ should not reveal any information about the private inputs to the client (i.e., A and $\mathbf{r}$) other than the underlying plaintext $A \cdot \mathbf{m} - \mathbf{r}$. Otherwise, some information is leaked. Gazelle proposes two methods to provide circuit privacy that are not incorporated in their implementation. Hence, we need to scale up their performance numbers for a fair comparison.

The first method is to let the client and server engage in a two-party secure decryption protocol, where the input of client is *sk* and input of server is $\mathbf{c}$. However, this method adds communication and needs extra rounds of interaction. A more widely used approach is *noise flooding*. Roughly speaking, the server adds a large noise term to $\mathbf{c}$ before returning it to the client. The noise is big enough to drown any extra information contained in the ciphertext, and still small enough to so that it still decrypts to the same plaintext.

For the concrete instantiation of Gazelle, one needs to triple the size of ciphertext modulus q from 60 bits to 180 bits, and increase the ring dimension n from 2048 to 8192. The (amortized) complexity of homomorphic operations in the BFV scheme is approximately $O(\log n \log q)$, with the exception that some operations run in $O(\log q)$ amortized time. Therefore, adding noise flooding would result in a *3-3.6 times slow down* for the HE component of Gazelle. To give some concrete examples, we consider two networks used for benchmarking in Gazelle: MNIST-D and CIFAR-10 networks. For the MNIST-D network, homomorphic encryption takes 55% and 22% in online and total time, respectively. For CIFAR-10, the corresponding figures are 35%, and 10%[1]. Therefore, we estimate that the total time for MNIST-D will grow from 0.81s to 1.16-1.27s (network BM3 in this chapter). In the case of CIFAR-10 network, the total time will grow from

[1] these percentage numbers are obtained through private communication with the authors.

12.9s to 15.48-16.25s.

4.4 The XONN Framework

In this section, we explain how neural networks can be trained such that they incur a minimal cost during the oblivious inference. The most computationally intensive operation in a neural network is matrix multiplication. In GC, each multiplication has a *quadratic* computation and communication cost with respect to the input bit-length. This is the major source of inefficiency in prior work [RRK18]. We overcome this limitation by changing the learning process such that the trained neural network's weights become binary. As a result, costly multiplication operations are replaced with `XNOR` gates which are essentially free in GC. We describe the training process in Section 4.4.1. In Section 4.4.2, we explain the operations and their corresponding Boolean circuit designs that enable a very fast oblivious inference. In Section 4.5, we elaborate on XONN implementation.

4.4.1 Customized Network Binarization

Numerical optimization algorithms minimize a specific cost function associated with neural networks. It is well-known that neural network training is a non-convex optimization, meaning that there exist many locally-optimum parameter configurations that result in similar inference accuracies. Among these parameter settings, there exist solutions where both neural network parameters and activation units are restricted to take binary values (i.e., either $+1$ or -1); these solutions are known as Binary Neural Netowrks (BNNs) [CHS$^+$16].

One major shortcoming of BNNs is their (often) low inference accuracy. In the machine learning community, several methods have been proposed to modify BNN functionality for accuracy enhancement [RORF16, GSK18, LZP17]. These methods are devised for *plaintext* execution of BNNs and are not efficient for oblivious inference with GC. We emphasize that, when

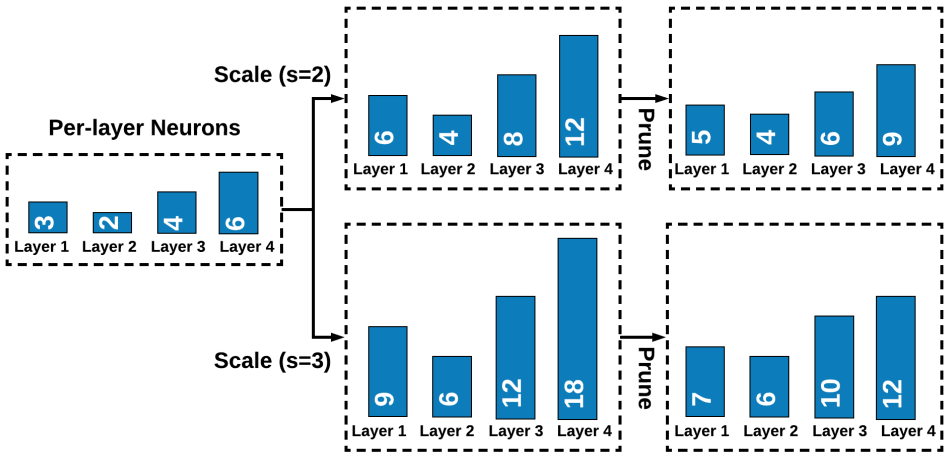


Figure 4.1: Illustration of BNN customization. The bars represent the number of neurons in each hidden layer.

modifying BNNs for accuracy enhancement, one should also take into account the implications in the corresponding GC circuit. With this in mind, we propose to modify the number of channels and neurons in CONV and FC layers, respectively. Increasing the number of channels/neurons leads to a higher accuracy but it also increases the complexity of the corresponding GC circuit. As a result, XONN provides a trade-off between the accuracy and the communication/runtime of the oblivious inference. This tradeoff enables cloud servers to customize the complexity of the GC protocol to optimally match the computation and communication requirements of the clients. To customize the BNN, XONN configures the per-layer number of neurons in two steps:

- Linear Scaling: Prior to training, we scale the number of channels/neurons in all BNN layers with the same factor (s), e.g., $s = 2$. Then, we train the scaled BNN architecture.
- Network Trimming: Once the (uniformly) scaled network is trained, a post-processing algorithm removes redundant channels/neurons from each hidden layer to reduce the GC cost while maintaining the inference accuracy.

Figure 4.1 illustrates the BNN customization method for an example baseline network with four hidden layers. Network trimming (pruning) consists of two steps, namely, Feature Ranking and Iterative Pruning which we describe next.

Feature Ranking: In order to perform network trimming, one needs to sort the channels/neurons of each layer based on their contribution to the inference accuracy. In conventional neural networks, simple ranking methods sort features based on absolute value of the neurons/channels [HPTD15]. In BNNs, however, the weights/features are either $+1$ or -1 and the absolute value is not informative. To overcome this issue, we utilize first order Taylor approximation of neural networks and sort the features based on the magnitude of the gradient values [MTK$^+$16]. Intuitively, the gradient with respect to a certain feature determines its importance; a high (absolute) gradient indicates that removing the neuron has a destructive effect on the inference accuracy. Inspired by this notion, we develop a feature ranking method described in Algorithm 2.

Iterative Pruning: We devise a step-by-step algorithm for model pruning which is summarized in Algorithm 3. At each step, the algorithm selects one of the BNN layers l^* and removes the first p^* features with the lowest importance (line 17). The selected layer l^* and the number of pruned neurons p^* maximize the following reward (line 15):

$$reward(l,p) = \frac{c_{curr} - c_{next}}{e^{a_{curr} - a_{next}}}, \tag{4.1}$$

where c_{curr} and c_{next} are the GC complexity of the BNN before and after pruning, whereas, a_{curr} and a_{next} denote the corresponding validation accuracies. The numerator of this reward encourages higher reduction in the GC cost while the denominator penalizes accuracy loss. Once the layer is pruned, the BNN is fine-tuned to recover the accuracy (line 18). The pruning process stops once the accuracy drops below a pre-defined threshold.

Algorithm 2 XONN Channel Sorting for CONV Layers

Input: Trained BNN with loss function $\mathcal{L}$, CONV layer l with output shape of $h1 \times h2 \times f$, subsampled validation data and labels $\{(\mathbf{X_1}, z_1), \ldots, (\mathbf{X_k}, z_k)\}$

Output: Indices of the sorted channels: $\{i_0, \ldots, i_f\}$

$\mathbf{G} \leftarrow zeros(k \times h1 \times h2 \times f)$ ▷ define gradient tensor

for $i = 1, \ldots, k$ **do**

  $\mathcal{L} = \mathcal{L}(\mathbf{X_i}, z_i)$ ▷ evaluate loss function

  $\nabla_Y = \frac{\partial \mathcal{L}}{\partial Y^l}$ ▷ compute gradient w.r.t. layer output

  $\mathbf{G}[i, :, :, :] \leftarrow \nabla_Y$ ▷ store gradient

end for

$\mathbf{G_{abs}} \leftarrow |\mathbf{G}|$ ▷ take elementwise absolute values

$\mathbf{g_s} \leftarrow zeros(f)$ ▷ define sum of absolute values

for $i = 1, \ldots, f$ **do**

  $\mathbf{g_s}[i] \leftarrow sum(\mathbf{G_{abs}}[:, :, :, i])$

end for

$\{i_0, \ldots, i_f\} \leftarrow sort(\mathbf{g_s})$

return $\{i_0, \ldots, i_f\}$

4.4.2 Oblivious Inference

BNNs are trained such that the weights and activations are binarized, i.e., they can only have two possible values: $+1$ or -1. This property allows BNN layers to be rendered using a simplified arithmetic. In this section, we describe the functionality of different layer types in BNNs and their Boolean circuit translations. Below, we explain each layer type.

Binary Linear Layer: Most of the computational complexity of neural networks is due to the linear operations in CONV and FC layers. As we discuss in Section 2.3, linear operations are realized using vector dot product (VDP). In BNNs, VDP operations can be implemented using simplified circuits. We categorize the VDP operations of this work into two classes: (i) Integer-VDP where only one of the vectors is binarized and the other has integer elements and (ii) Binary-VDP where both vectors have binary (± 1) values.

Algorithm 3 XONN Iterative BNN Pruning

Input: Trained BNN with n overall CONV and FC layers, minimum accuracy threshold θ, number of pruning trials per layer t, subsampled validation data and labels $data_V$, training data and labels $data_T$

Output: BNN with pruned layers

$\mathbf{p} \leftarrow zeros(n-1)$ ▷ current number of pruned neurons/channels per layer
$a_{curr} \leftarrow Accuracy(BNN, data_V|\mathbf{p})$ ▷ current BNN validation accuracy
$c_{curr} \leftarrow Cost(BNN|\mathbf{p})$ ▷ current GC cost
while $a_{curr} > \theta$ **do** ▷ repeat until accuracy drops below θ
  for $l = 1, \ldots, n-1$ **do** ▷ search over all layers
    $inds \leftarrow Rank(BNN, l, data_V)$ ▷ rank features via Algorithm 2
    $f \leftarrow$ Number of neurons/channels ▷ number of output neurons/channels
    for $p = \mathbf{p}[l], \mathbf{p}[l] + \frac{f}{t}, \ldots, f$ **do** ▷ search over possible pruning rates
      $BNN_{next} \leftarrow Prune(BNN, l, p, inds)$ ▷ prune p features with lowest ranks from the l-th layer
      $a_{next} \leftarrow Accuracy(BNN_{next}, data_V|\mathbf{p}[\mathbf{1}], \ldots, \mathbf{p}[\mathbf{l}] = p, \ldots, \mathbf{p}[\mathbf{n-1}])$ ▷ validation accuracy if pruned
      $c_{next} \leftarrow Cost(BNN_{next}|\mathbf{p}[\mathbf{1}], \ldots, \mathbf{p}[\mathbf{l}] = p, \ldots, \mathbf{p}[\mathbf{n-1}])$ ▷ GC cost if pruned
      $reward(l, p) = \frac{c_{curr} - c_{next}}{e^{(a_{curr} - a_{next})}}$ ▷ compute reward given that p features are pruned from layer l
    end for
  end for
  $\{l^*, p^*\} \leftarrow \arg\max_{l,p} reward(l,p)$ ▷ select layer l^* and pruning rate p^* that maximize the reward
  $\mathbf{p}[\mathbf{l}^*] \leftarrow p^*$ ▷ update the number of pruned features in vector $\mathbf{p}$
  $BNN \leftarrow Prune(BNN, l^*, p^*, inds)$ ▷ prune p^* features with lowest ranks from the l^*-th layer
  $BNN \leftarrow Fine\text{-}tune(BNN, data_T)$ ▷ fine-tune the pruned model using training data to recover accuracy
  $a_{curr} \leftarrow Accuracy(BNN, data_V|\mathbf{p})$ ▷ update current BNN validation accuracy
  $c_{curr} \leftarrow Cost(BNN|\mathbf{p})$ ▷ update current GC cost
end while
return BNN

Integer-VDP: For the first layer of the neural network, the server has no control over the input data which is not necessarily binarized. The server can only train binary weights and use them for oblivious inference. Consider an input vector $\mathbf{x} \in \mathbb{R}^n$ with integer (possibly fixed-point) elements and a weight vector $\mathbf{w} \in \{-1, 1\}^n$ with binary values. Since the elements of the binary vector can only take $+1$ or -1, the Integer-VDP can be rendered using additions and subtractions.

In particular, the binary weights can be used in a selection circuit that decides whether the pertinent integer input should be added to or subtracted from the VDP result.

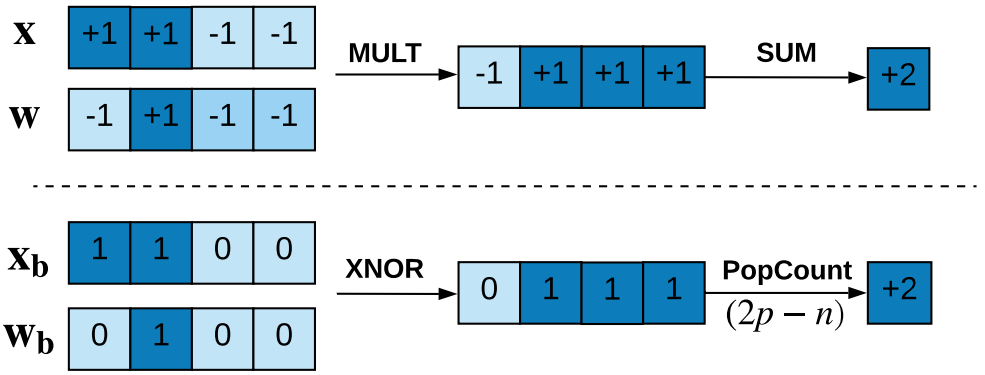


Figure 4.2: Equivalence of Binary-VDP and XnorPopcount.

Binary-VDP: Consider a dot product between two binary vectors $\mathbf{x} \in \{-1,+1\}^n$ and $\mathbf{w} \in \{-1,+1\}^n$. If we encode each element with one bit (i.e., $-1 \to 0$ and $+1 \to 1$), we obtain binary vectors $\mathbf{x}_b \in \{0,1\}^n$ and $\mathbf{w}_b \in \{0,1\}^n$. It has been shown that the dot product of $\mathbf{x}$ and $\mathbf{w}$ can be efficiently computed using an *XnorPopcount* operation [CHS+16]. Figure 4.2 depicts the equivalence of $\mathsf{VDP}(\mathbf{x},\mathbf{w})$ and $XnorPopcount(\mathbf{x}_b,\mathbf{w}_b)$ for a VDP between 4-dimensional vectors. First, element-wise XNOR operations are performed between the two binary encodings. Next, the number of set bits p is counted, and the output is computed as $2p-n$.

Binary Activation Function: A Binary Activation (BA) function takes input x and maps it to $y = Sign(x)$ where $Sign(\cdot)$ outputs either $+1$ or -1 based on the sign of its input. This functionality can simply be implemented by extracting the most significant bit of x.

Binary Batch Normalization: in BNNs, it is often useful to normalize feature x using a Batch Normalization (BN) layer before applying the binary activation function. More specifically, a BN layer followed by a BA is equivalent to:

$$y = Sign(\gamma \cdot x + \beta) = Sign(x + \frac{\beta}{\gamma}),$$

since γ is a positive value. The combination of the two layers (BN+BA) is realized by a comparison between x and $-\frac{\beta}{\gamma}$.

Binary Max-Pooling: Assuming the inputs to the max-pooling layers are binarized, taking the maximum in a window is equivalent to performing logical OR over the binary encodings as depicted in Figure 4.3. Note that average-pooling layers are usually not used in BNNs since the average of multiple binary elements is no longer a binary value.

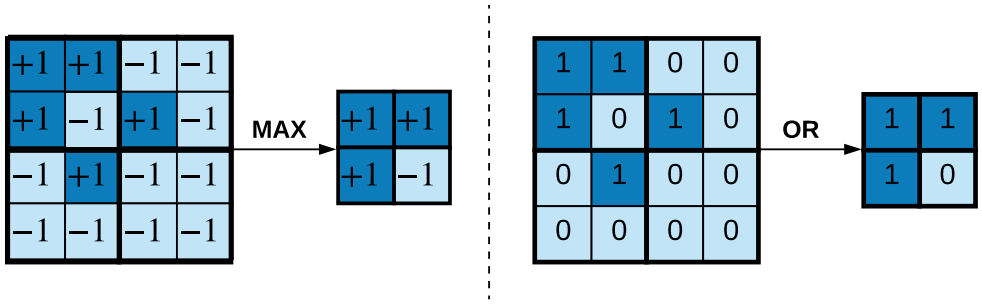


Figure 4.3: The equivalence between Max-Pooling and Boolean-OR operations in BNNs.

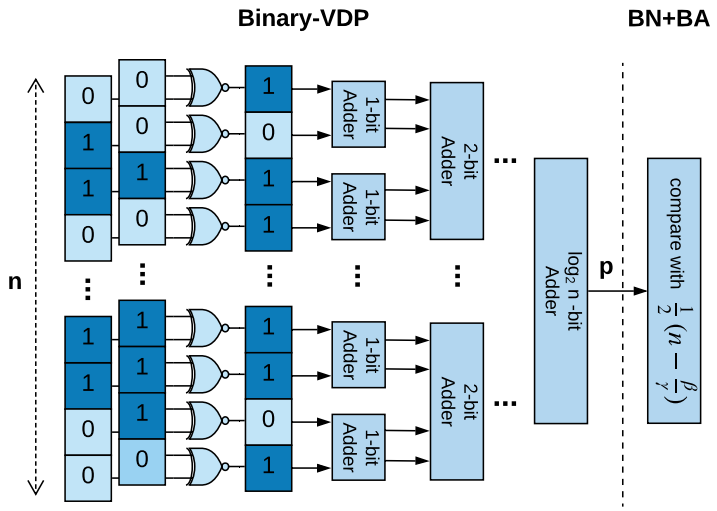


Figure 4.4: Circuit for binary-VDP followed by comparison for batch normalization (BN) and binary activation (BA).

Figure 4.4 demonstrates the Boolean circuit for Binary-VDP followed by BN and BA. The number of non-XOR gates for binary-VDP is equal to the number of gates required to render the tree-adder structure in Figure 4.4. Similarly, Figure 4.5 shows the Integer-VDP counterpart. In the first level of the tree-adder of Integer-VDP (Figure 4.5), the binary weights determine

whether the integer input should be added to or subtracted from the final result within the "Select" circuit. The next levels of the tree-adder compute the result of the integer-VDP using "Adder" blocks. The combination of BN and BA is implemented using a single *comparator*. Compared to Binary-VDP, Integer-VDP has a high garbling cost which is linear with respect to the number of bits. To mitigate this problem, we propose an alternative solution based on Oblivious Transfer (OT) in Section 4.4.3.

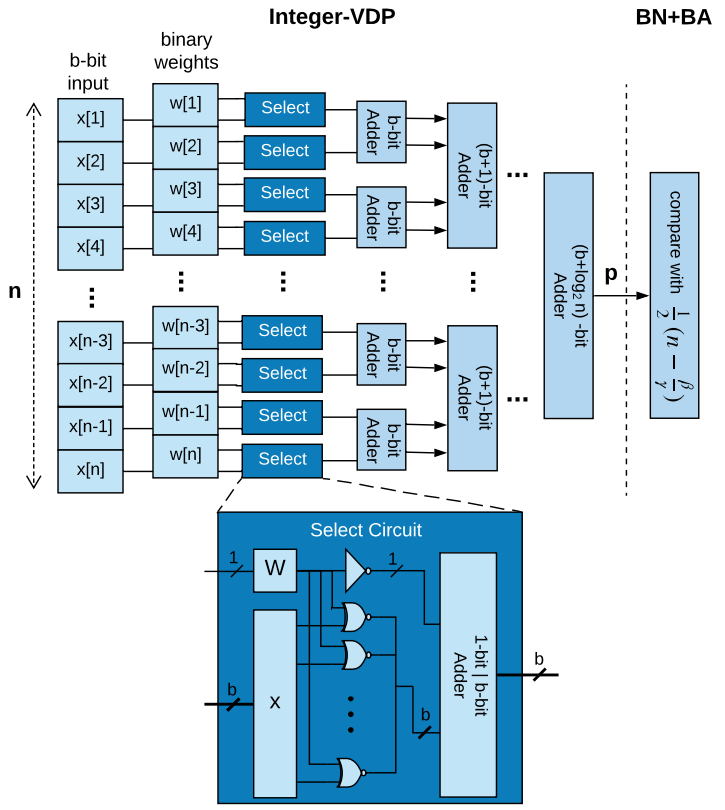


Figure 4.5: Circuit for Integer-VDP followed by comparison for batch normalization (BN) and binary activation (BN).

4.4.3 Oblivious Conditional Addition Protocol

In XONN, all of the activation values as well as neural network weights are binary. However, the input to the neural network is provided by the user and is not necessarily binary. The first layer of a typical neural network comprises either an FC or a CONV layer, both of which are

Sender:

(1) Bit-extends all elements of $\mathbf{v_1}$ and creates $\mathbf{v_1^*}$

(2) Creates two's complement of $\mathbf{v_1^*}$: $\overline{\mathbf{v_1^*}}$

(3) Creates random vector $\mathbf{r}$: same size as $\overline{\mathbf{v_1^*}}$

(4) Creates list of first messages as $\mathbf{m^2} = \overline{\mathbf{v_1^*}} - \mathbf{r}$ *mod* $2^{b'}$

(5) Creates list of second messages as $\mathbf{m^1} = \mathbf{v_1^*} - \mathbf{r}$ *mod* $2^{b'}$

Sender & Receiver:

(6) Parties engage in Oblivious Transfer (OT)
Sender puts $\mathbf{m^1}$ and $\mathbf{m^2}$ as message vectors
Receiver puts $\mathbf{v_2}$ vector as selection bits

Receiver:

(7) Gets vector $\mathbf{v_t}$ where:

$$\mathbf{v_t}[i] = \begin{cases} \overline{\mathbf{v_1^*}}[i] - \mathbf{r}[i] \ mod \ 2^{b'} \ (\text{if } \mathbf{v_2}[i] = 0) \\ \mathbf{v_1^*}[i] - \mathbf{r}[i] \ mod \ 2^{b'} \ (\text{if } \mathbf{v_2}[i] = 1) \end{cases}$$

Sender:

(8) Computes her additive share of VDP result as:

$$y_1 = \sum_{i=1}^{n} \mathbf{r}[i] \ mod \ 2^{b'}$$

Receiver:

(9) Computes his additive share of VDP result as:

$$y_2 = \sum_{i=1}^{n} \mathbf{v_t}[i] \ mod \ 2^{b'}$$

Figure 4.6: Oblivious Conditional Addition (OCA) protocol.

evaluated using oblivious Integer-VDP. On the one side, the user provides her input as non-binary (integer) values. On the other side, the network parameters are binary values representing -1 and 1. We now demonstrate how Integer-VDP can be described as an OT problem. Let us denote the user's input as a vector $\mathbf{v_1}$ of n (b-bit) integers. The server holds a vector of n binary values denoted by $\mathbf{v_2}$. The result of Integer-VDP is a number "y" that can be described with

$$b' = \lceil * \rceil \log_2(n \cdot (2^b - 1))$$

bits. Figure 4.6 summarizes the steps in the OCA protocol. The first step is to *bit-extend* $\mathbf{v_1}$ from b-bit to b'-bit. In other words, if $\mathbf{v_1}$ is a vector of *signed* integer/fixed-point numbers, the most significant bit should be repeated $(b' - b)$-many times, otherwise, it has to be zero-padded for

most significant bits. We denote the bit-extended vector by $\mathbf{v_1^*}$. The second step is to create the two's complement vector of $\mathbf{v_1^*}$, called $\overline{\mathbf{v_1^*}}$. The client also creates a vector of n (b'-bit) randomly generated numbers, denoted as $\mathbf{r}$. She computes element-wise vector subtractions $\mathbf{v_1^*} - \mathbf{r}$ *mod* $2^{b'}$ and $\overline{\mathbf{v_1^*}} - \mathbf{r}$ *mod* $2^{b'}$. These two vectors are n-many pair of messages that will be used as input to n-many 1-out-of-two OTs. More precisely, $\overline{\mathbf{v_1^*}} - \mathbf{r}$ *mod* $2^{b'}$ is a list of first messages and $\mathbf{v_1^*} - \mathbf{r}$ *mod* $2^{b'}$ is a list of second messages. The server's list of selection bits is $\mathbf{v_2}$. After n-many OTs are finished, the server has a list of n transferred numbers called $\mathbf{v_t}$ where

$$\mathbf{v_t}[i] = \begin{cases} \overline{\mathbf{v_1^*}}[i] - \mathbf{r}[i] \ mod \ 2^{b'} & if \ \ \mathbf{v_2}[i] = 0 \\ \mathbf{v_1^*}[i] - \mathbf{r}[i] \ mod \ 2^{b'} & if \ \ \mathbf{v_2}[i] = 1 \end{cases} \quad i = 1, \ \dots \ , \ n.$$

Finally, the client computes $y_1 = \sum_{i=1}^{n} \mathbf{r}[i] \ mod \ 2^{b'}$ and the server computes $y_2 = \sum_{i=1}^{n} \mathbf{v_t}[i] \ mod \ 2^{b'}$. By OT's definition, the receiver (server) gets only one of the two messages from the sender. That is, based on each selection bit (a binary weight), the receiver gets an additive share of either the sender's number or its two's complement. Upon adding all of the received numbers, the receiver computes an additive share of the Integer-VDP result. Now, even though the sender does not know which messages were selected by the receiver, she can add all of the randomly generated numbers $\mathbf{r}[i]$s which is equal to the other additive share of the Integer-VDP result. Since all numbers are described in the two's complement format, subtractions are equivalent to the addition of the two's complement values, which are created by the sender at the beginning of OCA. Moreover, it is possible that as we accumulate the values, the bit-length of the final Integer-VDP result grows accordingly. This is supported due to the bit-extension process at the beginning of the protocol. In other words, all additions are performed in a larger ring such that the result does not overflow.

Note that all numbers belong to the ring $\mathbb{Z}_{2^{b'}}$ and by definition, a ring is closed under addition, therefore, y_1 and y_2 are true additive shares of $y = y_1 + y_2 \ mod \ 2^{b'}$. We described the OCA protocol for one Integer-VDP computation. As we outlined in Section 4.4.2, all linear

operations in the first layer of the DL model (either FC or CONV) can be formulated as a series of Integer-VDPs.

In traditional OT, public-key encryption is needed for each OT invocation which can be computationally expensive. Thanks to the Oblivious Transfer Extension technique [IKNP03, Bea96, ALSZ13], one can perform many OTs using symmetric-key encryption and only a fixed number of public-key operations.

Required Modification to the Next Layer. So far, we have shown how to perform Integer-VDP using OT. However, we need to add an "addition" layer to reconstruct the true value of y from its additive shares before further processing it. The overhead of this layer, as well as OT computations, are discussed next. Note that OCA is used only for the first layer and it does not change the overall constant round complexity of XONN since it is performed only once regardless of the number of layers in the DL model.

Comparison to Integer-VDP in GC. Table 4.2 shows the computation and communication costs for two approaches: (i) computing the first layer in GC and (ii) utilizing OCA. OCA removes the GC cost of the first layer in XONN. However, it adds the overhead of a set of OTs and the GC costs associated with the new ADD layer.

Table 4.2: Computation and communication cost of OCA.

Costs {Sender, Receiver}	**GC**	**OCA**	
		OT	ADD Layer
Comp. (AES ops)	$(n+1)\cdot b\cdot\{2,4\}$	$n\cdot\{1,2\}$	$b'\cdot\{2,4\}$
Comm. (bit)	$(n+1)\cdot b\cdot 2\cdot 128$	$n\cdot b$	$b'\cdot 2\cdot 128$

4.4.4 Security of XONN

We consider the Honest-but-Curious (HbC) adversary model consistent with all of the state-of-the-art solutions for oblivious inference [RRK18, MZ17b, LJLA17b, RWT+18a, CGR+17, JVC18]. In this model, neither of the involved parties is trusted but they are assumed to follow the

protocol. Both server and client cannot infer any information about the other party's input from the entire protocol transcript. XONN relies solely on the GC and OT protocols, both of which are proven to be secure in the HbC adversary model in [LP09] and [Rab05], respectively. Utilizing binary neural networks does not affect GC and OT protocols in any way. More precisely, we have changed the function $f(.)$ that is evaluated in GC such that it is more efficient for the GC protocol: drastically reducing the number of `AND` gates and using `XOR` gates instead. Our novel Oblivious Conditional Addition (OCA) protocol (Section 4.4.3) is also based on the OT protocol. The sender creates a list of message pairs and puts them as input to the OT protocol. Each message is an additive share of the sender's private data from which the secret data cannot be reconstructed. The receiver puts a list of selection bits as input to the OT. By OT's definition, the receiver learns nothing about the unselected messages and the sender does not learn the selection bits.

During the past few years, several attacks have been proposed that extract some information about the DL model by querying the server many times [TZJ+16, FJR15, SSSS17]. It has been shown that some of these attacks can be effective in the black-box setting where the client only receives the prediction results and does not have access to the model. Therefore, considering the definition of an oblivious inference, these type of attacks are out of the scope of oblivious inference frameworks. However, in Appendix 4.6, we show how these attacks can be thwarted by adding a simple layer at the end of the neural network which adds a negligible overhead.

Security Against Malicious Adversaries. The HbC adversary model is the standard security model in the literature. However, there are more powerful security models such as security against covert and malicious adversaries. In the malicious security model, the adversary (either the client or server) can deviate from the protocol at any time with the goal of learning more about the input from the other party. One of the main distinctions between XONN and the state-of-the-art solutions is that XONN can be automatically adapted to the malicious security using cut-and-choose techniques [LP12, HKE13, Lin16]. These methods take a GC protocol in HbC and readily extend it to the malicious security model. This modification increases the overhead

but enables a higher security level. To the best of our knowledge, there is no practical solution to extend the customized mixed-protocol frameworks [LJLA17b, RWT+18a, CGR+17, JVC18] to the malicious security model. Our GC-based solution is more efficient compared to the mixed-protocol solutions and can be upgraded to the malicious security at the same time.

4.5 The XONN Implementation

In this section, we elaborate on the garbling/evaluation implementation of XONN. All of the optimizations and techniques proposed in this section do not change the security or correctness in anyway and only enable the framework's scalability for large network architectures.

We design a new GC framework with the following design principles in mind: (i) *Efficiency:* XONN is designed to have a minimal data movement and low cache-miss rate. (ii) *Scalability:* oblivious inference inevitably requires significantly higher memory usage compared to plaintext evaluation of neural networks. High memory usage is one critical shortcoming of state-of-the-art secure computation frameworks. As we show in our experimental results, XONN is designed to scale for very deep neural networks that have higher accuracy compared to networks considered in prior art. (iii) *Modularity:* our framework enables users to create Boolean description of different layers separately. This allows the hardware synthesis tool to generate more optimized circuits as we discuss in Section 4.5.1. (iv) *Ease-to-use:* XONN provides a very simple API that requires few lines of neural network description. Moreover, we have created a compiler that takes a Keras description and automatically creates the network description for XONN API.

XONN is written in C++ and supports all major GC optimizations proposed previously. Since the introduction of GC, many optimizations have been proposed to reduce the computation and communication complexity of this protocol. Bellare et al. [BHKR13] have provided a way to perform garbling using efficient fixed-key AES encryption. Our implementation benefits from this

optimization by using Intel AES-NI instructions. Row-reduction technique [NPS99] reduces the number of garbled tables from four to three. Half-Gates technique [ZRE15] further reduces the number of rows in the garbled tables from three to two. One of the most influential optimizations for the GC protocol is the *free-XOR* technique [KS08a] which makes `XOR`, `XNOR`, and `NOT` almost free of cost. Our implementation for Oblivious Transfer (OT) is based on libOTe [Rin].

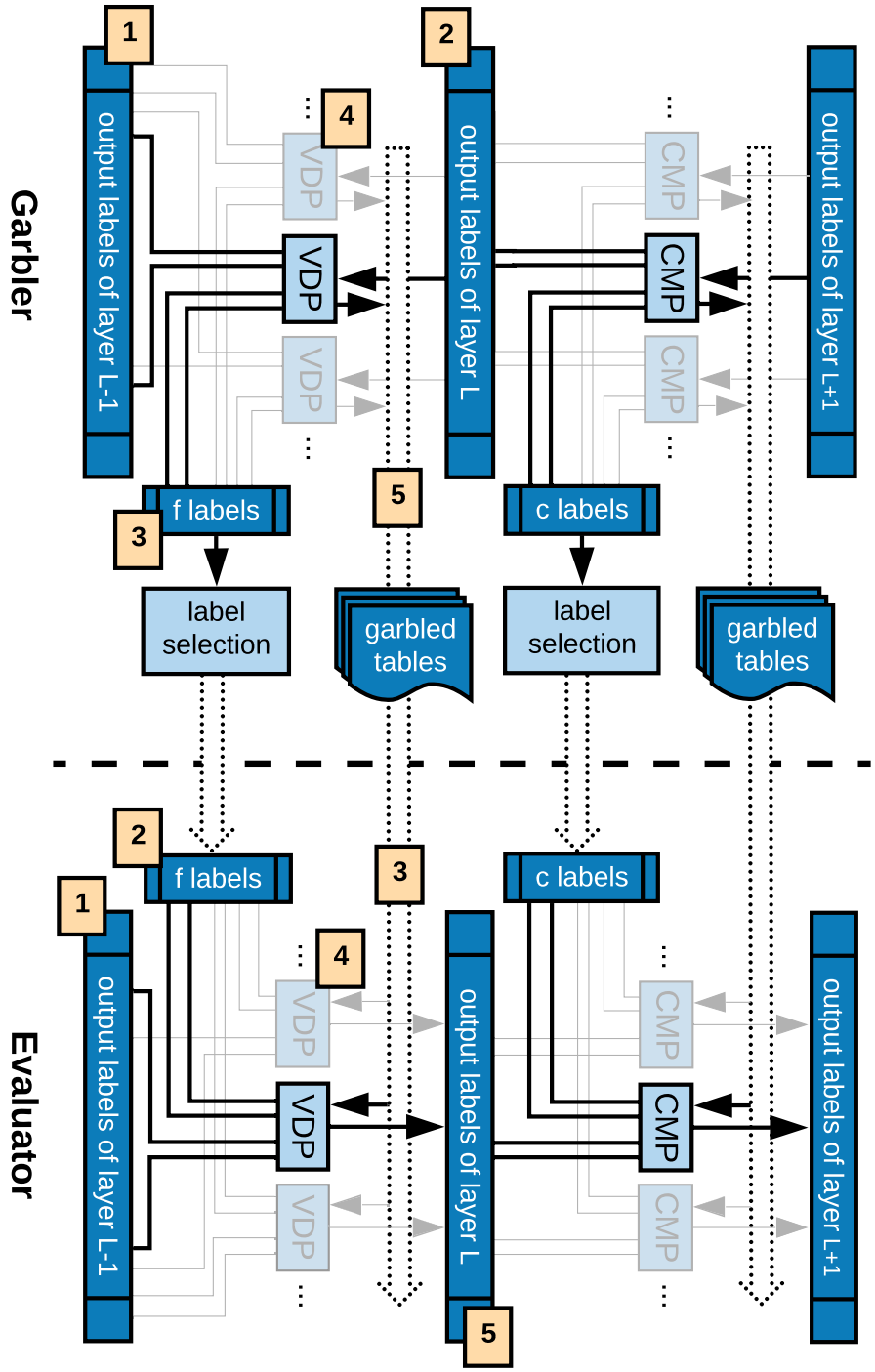


Figure 4.7: XONN modular and pipelined garbling engine.

4.5.1 Modular Circuit Synthesis and Garbling

In XONN, each layer is described as multiple invocations of a *base* circuit. For instance, linear layers (CONV and FC) are described by a VDP circuit. MaxPool is described by an OR circuit where the number of inputs is the window size of the MaxPool layer. BA/BN layers are described using a comparison (CMP) circuit. The memory footprint is significantly reduced in this approach: we only create and store the base circuits. As a result, the connection between two invocations of two different base circuits is handled at the software level.

We create the Boolean circuits using TinyGarble [SHS+15] hardware synthesis approach. TinyGarble's technology libraries are optimized for GC and produce circuits that have low number of non-XOR gates. Note that the Boolean circuit description of the contemporary neural networks comprises between millions to billions of Boolean gates, whereas, synthesis tools cannot support circuits of this size. However, due to XONN modular design, one can synthesize each base circuit separately. Thus, the bottleneck transfers from the synthesis tool's maximum number of gates to the system's memory. As such, XONN effectively scales for any neural network complexity regardless of the limitations of the synthesis tool as long as enough memory (i.e., RAM) is available. Later in this section, we discuss how to increase the scalability by dynamically managing the allocated memory.

Pipelined GC Engine. In XONN, computation and communication are pipelined. For instance, consider a CONV layer followed by an activation layer. We garble/evaluate these layers by multiple invocations of the VDP and CMP circuits (one invocation per output neuron) as illustrated in Figure 4.7. Upon finishing the garbling process of layer $L-1$, the Garbler starts garbling the L^{th} layer and creates the random labels for output wires of layer L. He also needs to create the random labels associated with his input (i.e., the weight parameters) to layer L. Given a set of input and output labels, Garbler generates the garbled tables, and sends them to the Evaluator as soon as one is ready. He also sends one of the two input labels for his input bits. At the same time, the Evaluator has computed the output labels of the $(L-1)^{th}$ layer. She receives

the garbled tables as well as the Garbler's selected input labels and decrypts the tables and stores the output labels of layer L.

Dynamic Memory Management. We design the framework such that the allocated memory for the labels is released as soon as it is no longer needed, reducing the memory usage significantly. For example, without our dynamic memory management, the Garbler had to allocate 10.41GB for the labels and garbled tables for the entire garbling of BC1 network (see Section 4.7 for network description). In contrast, in our framework, the size of memory allocation never exceeds 2GB and is less than 0.5GB for most of the layers.

4.5.2 Application Programming Interface (API)

XONN provides a simplified and easy-to-use API for oblivious inference. The framework accepts a high-level description of the network, parameters of each layer, and input structure. It automatically computes the number of invocations and the interconnection between all of the base circuits. Figure 4.8 shows the complete network description that a user needs to write for a sample network architecture (the BM3 architecture, see Section 4.7). All of the required circuits are automatically generated using TinyGarble [SHS+15] synthesis libraries. It is worth mentioning that for the task of oblivious inference, our API is much simpler compared to the recent *high-level* EzPC framework [CGR+17]. For example, the required lines of code to describe BM1, BM2, and BM3 network architectures (see Section 4.7) in EzPC are 78, 88, and 154, respectively. In contrast, they can be described with only 6, 6, and 10 lines of code in our framework.

```
1  INPUT 28 1 8
2  CONV 5 16 1 0 OCA
3  ACT
4  MAXPOOL 2
5  CONV 5 16 1 0
6  ACT
7  MAXPOOL 2
8  FC 100
9  ACT
10 FC 10
```

```
Description:

INPUT #input_feature #channels #bit-length

CONV #filter_size #filters #stride
       #Pad #OCA (optional)

MAXPOOl #window_size

FC #output_neurons
```

Figure 4.8: Sample snippet code in XONN.

Keras to XONN Translation. To further facilitate the adaptation of XONN, a compiler is created to translate the description of the neural network in Keras [Cho15] to the XONN format. The compiler creates the `.xonn` file and puts the network parameters into the required format (`HEX` string) to be read by the framework during the execution of the GC protocol. All of the parameter adjustments are also automatically performed by the compiler.

4.6 Attacks on Deep Neural Networks

In this section, we review three of the most important attacks against deep neural networks that are relevant to the context of oblivious inference [TZJ+16, FJR15, SSSS17]. In all three, a client-server model is considered where the client is the adversary and attempts to learn more about the model held by the server. The client sends many inputs and receives the inference results . He then analyzes the results to infer more information about either the network parameters or the training data that has been used in the training phase of the model. We briefly review each attack and illustrate a simple defense mechanism with negligible overhead based on the suggestions provided in these works.

Model Inversion Attack [FJR15]. In the black-box access model of this attack (which fits the computational model of this work), an adversarial client attempts to learn about a prototypical sample of one of the classes. The client iteratively creates an input that maximizes the confidence score corresponding to the target class. Regardless of the specific training process, the attacker can learn significant information by querying the model many times.

Model Extraction Attack [TZJ+16]. In this type of attack, an adversary's goal is to estimate the parameters of the machine learning model held by the server. For example, in a logistic regression model with n parameters, the model can be extracted by querying the server n times and upon receiving the confidence values, solving a system of n equations. Model extraction can diminish the pay-per-prediction business model of technology companies. Moreover, it can

be used as a pre-step towards the model inversion attack.

Membership Inference Attack [SSSS17]. The objective of this attack is to identify whether a given input has been used in the training phase of the model or not. This attack raises certain privacy concerns. The idea behind this attack is that the neural networks usually perform better on the data that they were trained on. Therefore, two inputs that belong to the same class, one used in the training phase and one not, will have noticeable differences in the confidence values. This behavior is called *overfitting*. The attack can be mitigated using regularization techniques that reduce the dependency of the DL model on a single training sample. However, overfitting is not the only contributor to this information leakage.

Defense Mechanisms. In the prior state-of-the-art oblivious inference solution [LJLA17b], it has been suggested to limit the number of queries from a specific client to limit the information leakage. However, in practice, an attacker can impersonate himself as many different clients and circumvent this defense mechanism. Note that all three attacks rely on the fact that along with the inference result, the server provides the confidence vector that specifies how likely the client's input belongs to each class. Therefore, as suggested by prior work [TZJ+16, FJR15, SSSS17], it is recommended to augment a *filter* layer that (i) rounds the confidence scores or (ii) selects the index of a class that has the highest confidence score.

1. *Rounding the confidence values:* Rounding the values simply means omitting one (or more) of the Least Significant Bit (LSB) of all of the numbers in the last layer. This operation is in fact *free* in GC since it means Garbler has to avoid providing the mapping for those LSBs.
2. *Reporting the class label:* This operation is equivalent to computing `argmax` on the last layer. For a vector of size c where each number is represented with b bits, `argmax` is translated to $c \cdot (2b+1)$ many non-`XOR` (`AND`) gates. For example, in a typical architecture for MNIST (e.g., BM3) or CIFAR-10 dataset (e.g., BC1), the overhead is 1.68E-2% and 1.36E-4%, respectively.

Note that the two aforementioned defense mechanisms can be augmented to any framework

that supports non-linear functionalities [LJLA17b, RWT+18a, RRK18]. However, we want to emphasize that compared to mixed-protocol solutions, this means that another round of communication is usually needed to support the filter layer. Whereas, in XONN the filter layer does not increase the number of rounds and has negligible overhead compared to the overall protocol.

4.7 Experimental Results

We evaluate XONN on MNIST and CIFAR10 datasets, which are two popular classification benchmarks used in prior work. In addition, we provide four healthcare datasets to illustrate the applicability of XONN in real-world scenarios. For training XONN, we use Keras [Cho15] with Tensorflow backend [ABC+16]. The source code of XONN is compiled with GCC 5.5.0 using O3 optimization. All Boolean circuits are synthesized using Synopsys Design Compiler 2015. Evaluations are performed on (Ubuntu 16.04 LTS) machines with Intel-Core i7-7700k and 32GB of RAM. The experimental setup is comparable (but has less computational power) compared to the prior art [JVC18]. Consistent with prior frameworks, we evaluate the benchmarks in the LAN setting.

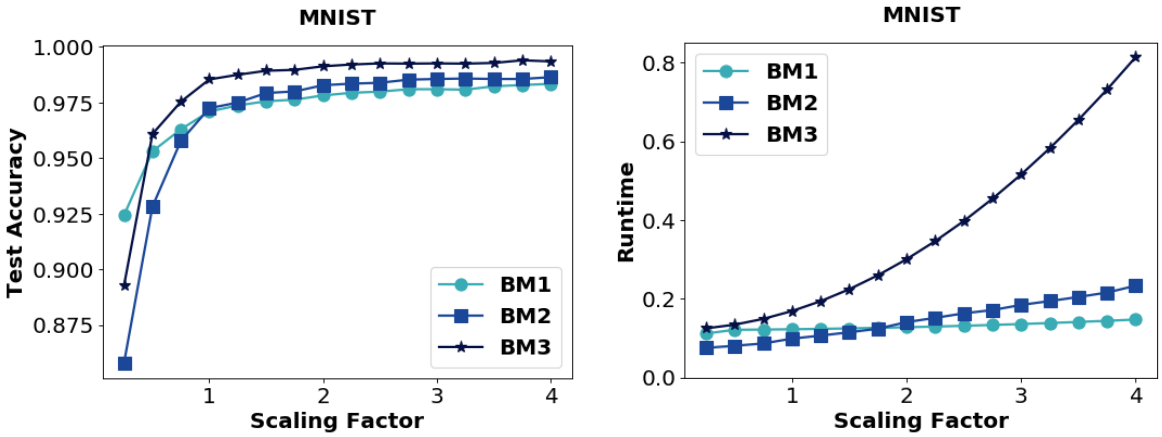
4.7.1 Evaluation on MNIST

There are mainly three network architectures that prior works have implemented for the MNIST dataset. We convert these reference networks into their binary counterparts and train them using the standard BNN training algorithm [CHS+16]. Table 4.3 summarizes the architectures for the MNIST dataset.

Table 4.3: Summary of the trained binary network architectures evaluated on the MNIST dataset. Detailed descriptions are available in Appendix A, Table A.1.

Arch.	Previous Papers	Description
BM1	SecureML [MZ17b], MiniONN [LJLA17b]	3 FC
BM2	CryptoNets [DGBL+16b], MiniONN [LJLA17b], DeepSecure [RRK18], Chameleon [RWT+18a]	1 CONV, 2 FC
BM3	MiniONN [LJLA17b], EzPC [CGR+17]	2 CONV, 2MP, 2FC

Analysis of Network Scaling: Recall that the classification accuracy of XONN is controlled by scaling the number of neurons in all layers (Section 4.4.1). Figure 4.9a depicts the inference accuracy with different scaling factors (more details in Table 4.11 in Section 4.7.5). As we increase the scaling factor, the accuracy of the network increases. This accuracy improvement comes at the cost of a higher computational complexity of the (scaled) network. As a result, increasing the scaling factor leads to a higher runtime. Figure 4.9b depicts the runtime of different BNN architectures as a function of the scaling factor s. Note that the runtime grows (almost) quadratically with the scaling factor due to the quadratic increase in the number of *Popcount* operations in the neural network (see $BM3$). However, for the $BM1$ and $BM2$ networks, the overall runtime is dominated by the constant initialization cost of the OT protocol ($\sim$ 70 millisecond).



(a) (b)

Figure 4.9: Effect of scaling factor on (a) accuracy and (b) inference runtime of MNIST networks. No pruning was applied in this evaluation.

GC Cost and the Effect of OCA: The communication cost of GC is the key contributor to the overall runtime of XONN. Here, we analyze the effect of the scaling factor on the total message size. Figure 4.10 shows the communication cost of GC for the BM1 and BM2 network architectures. As can be seen, the message size increases with the scaling factor. We also observe that the OCA protocol drastically reduces the message size. This is due to the fact that the first layer of BM1 and BM2 models account for a large portion of the overall computation; hence, improving the first layer with OCA has a drastic effect on the overall communication.

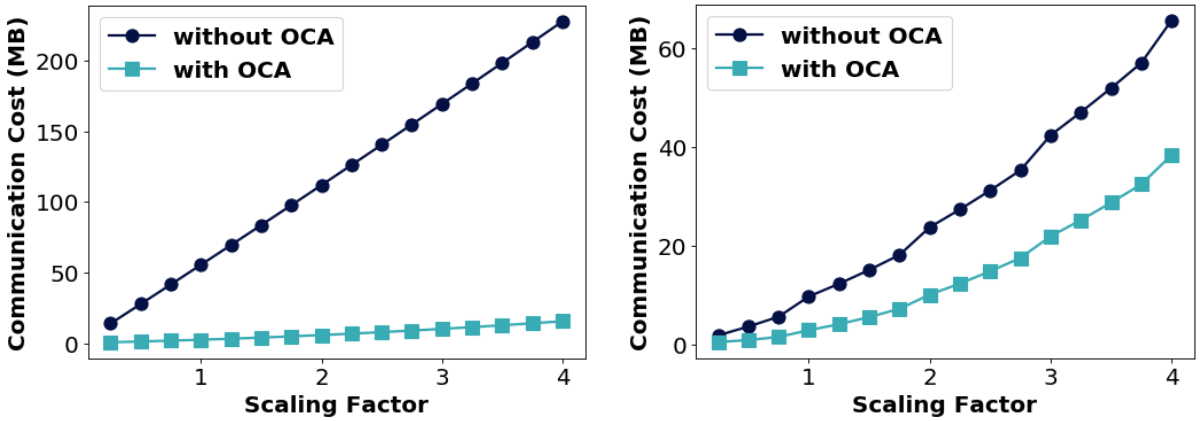


Figure 4.10: Effect of OCA on the communication of the BM1 (left) and BM2 (right) networks for different scaling factors. No pruning was applied in this evaluation.

Comparison to Prior Art: We emphasize that, unlike previous work, the accuracy of XONN can be customized by tuning the scaling factor (s). Furthermore, our channel/neuron pruning step (Algorithm 3) can reduce the GC cost in a post-processing phase. To provide a fair comparison between XONN and prior art, we choose a proper scaling factor and trim the pertinent scaled BNN such that the corresponding BNN achieves the same accuracy as the previous work. Table 4.4 compares XONN with the previous work in terms of accuracy, latency, and communication cost (a.k.a., message size). The last column shows the scaling factor (s) used to increase the width of the hidden layers of the BNN. Note that the scaled network is further trimmed using Algorithm 3.

In XONN, the runtime for oblivious transfer is at least ~ 0.07 second for initiating the

protocol and then it grows linearly with the size of the garbled tables; As a result, in very small architectures such as *BM*1, our solution is slightly slower than previous works since the constant runtime dominates the total runtime. However, for the *BM*3 network which has higher complexity than *BM*1 and *BM*2, XONN achieves a more prominent advantage over prior art. In summary, our solution achieves up to 7.7× faster inference (average of 3.4×) compared to Gazelle [JVC18]. Compared to MiniONN [LJLA17b], XONN has up to 62× lower latency (average of 26×) Table 4.4. Compared to EzPC [CGR$^+$17], our framework is 34× faster. XONN achieves 37.5×, 1859×, 60.4×, and 14× better latency compared to SecureML [MZ17b], CryptoNets [DGBL$^+$16b], DeepSecure [RRK18], and Chameleon [RWT$^+$18a], respectively.

Table 4.4: Comparison of XONN with the state-of-the-art for the MNIST network architectures.

Arch.	**Framework**	**Runtime (s)**	**Comm. (MB)**	**Acc.** (%)	**s**
BM1	SecureML	4.88	-	93.1	-
	MiniONN	1.04	15.8	97.6	-
	EzPC	0.7	76	97.6	-
	Gazelle	0.09	0.5	97.6	-
	XONN	0.13	4.29	97.6	1.75
BM2	CryptoNets	297.5	372.2	98.95	-
	DeepSecure	9.67	791	98.95	-
	MiniONN	1.28	47.6	98.95	-
	Chameleon	2.24	10.5	99.0	-
	EzPC	0.6	70	99.0	-
	Gazelle	0.29	8.0	99.0	-
	XONN	0.16	38.28	98.64	4.00
BM3	MiniONN	9.32	657.5	99.0	-
	EzPC	5.1	501	99.0	-
	Gazelle	1.16	70	99.0	-
	XONN	0.15	32.13	99.0	2.00

4.7.2 Evaluation on CIFAR-10

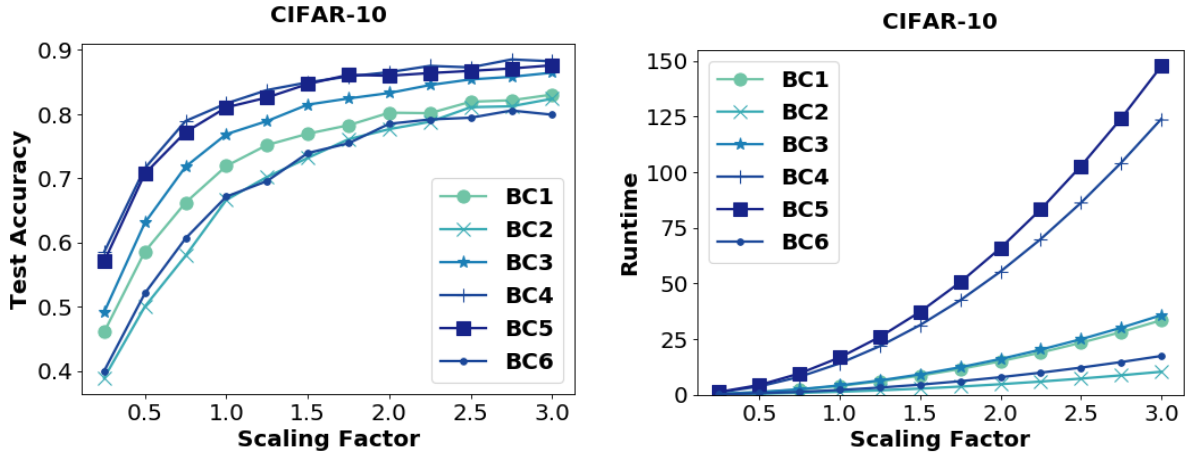
In Tables A.2- A.6, we summarize the network architectures that we use for the CIFAR-10 dataset. In this table, BC1 is the binarized version of the architecture proposed by MiniONN. To evaluate the scalability of our framework to larger networks, we also binarize the Fitnet [RBK$^+$14]

architectures, which are denoted as BC2-BC5. We also evaluate XONN on the popular VGG16 network architecture (BC6). Detailed architecture descriptions are available in Section 4.7.5, Table A.6.

Table 4.5: Summary of the trained binary network architectures evaluated on the CIFAR-10 dataset.

Arch.	Previous Papers	Description
BC1	MiniONN [LJLA17b], Chameleon [RWT+18a], EzPC [CGR+17], Gazelle [JVC18]	7 CONV, 2 MP, 1 FC
BC2	Fitnet [RBK+14]	9 CONV, 3 MP, 1 FC
BC3	Fitnet [RBK+14]	9 CONV, 3 MP, 1 FC
BC4	Fitnet [RBK+14]	11 CONV, 3 MP, 1 FC
BC5	Fitnet [RBK+14]	17 CONV, 3 MP, 1 FC
BC6	VGG16 [SZ14]	13 CONV, 5 MP, 3 FC

Analysis of Network Scaling: Similar to the analysis on the MNIST dataset, we show that the accuracy of our binary models for CIFAR-10 can be tuned based on the scaling factor that determines the number of neurons in each layer. Figure 4.11a depicts the accuracy of the BNNs with different scaling factors. As can be seen, increasing the scaling factor enhances the classification accuracy of the BNN. The runtime also increases with the scaling factor as shown in Figure 4.11b (more details in Table 4.12, Section 4.7.5).



(a) (b)

Figure 4.11: (a) Effect of scaling factor on accuracy for CIFAR-10 networks. (b) Effect of scaling factor on runtime. No pruning was applied in this evaluation.

Comparison to Prior Art: We scale the BC2 network with a factor of $s = 3$, then prune it using Algorithm 3. Details of pruning steps are available in Table 4.10 in Section 4.7.4. The resulting network is compared against prior art in Table 4.6. As can be seen, our solution achieves 2.7×, 45.8×, 9.1×, and 93.1× lower latency compared to Gazelle, EzPC, Chameleon, and MiniONN, respectively.

Table 4.6: Comparison of XONN with prior art on CIFAR-10.

Framework	Runtime (s)	Comm. (MB)	Acc. (%)	s
MiniONN	544	9272	81.61	-
Chameleon	52.67	2650	81.61	-
EzPC	265.6	40683	81.61	-
Gazelle	15.48	1236	81.61	-
XONN	5.79	2599	81.85	3.00

4.7.3 Evaluation on Medical Datasets

One of the most important applications of oblivious inference is medical data analysis. Recent advances in deep learning greatly benefit many complex diagnosis tasks that require exhaustive manual inspection by human experts [ERR+19, EKN+17, ADWF15, ROC+18]. To showcase the applicability of oblivious inference in real-world medical applications, we provide several benchmarks for publicly available healthcare datasets summarized in Table 4.7. We split the datasets into validation and training portions as indicated in the last two columns of Table 4.7. All datasets except Malaria Infection are normalized to have 0 mean and standard deviation of 1 per feature. The images of Malaria Infection dataset are resized to 32×32 pictures. The normalized datasets are quantized up to 3 decimal digits. Detailed architectures are available in Appendix A, Table A.7 We report the validation accuracy along with inference time and message size in Table 4.8.

Table 4.7: Summary of medical application benchmarks.

Task	Arch.	Description	# of Samples	
			Tr.	Val.
Breast Cancer [bre]	BH1	3 FC	453	113
Diabetes [dia]	BH2	3 FC	615	153
Liver Disease [liv]	BH3	3 FC	467	116
Malaria Infection [mal]	BH4	2 CONV, 2 MP, 2 FC	24804	2756

Table 4.8: Runtime, communication cost (Comm.), and accuracy (Acc.) for medical benchmarks.

Arch.	Runtime (ms)	Comm. (MB)	Acc. (%)
BH1	82	0.35	97.35
BH2	75	0.16	80.39
BH3	81	0.3	80.17
BH4	482	120.75	95.03

4.7.4 Network Trimming Examples

Table 4.9 and 4.10 summarize the trimming steps for the MNIST and CIFAR-10 benchmarks, respectively.

Table 4.9: Trimming MNIST architectures.

Network	Property	Trimming Step				Change
		initial	step 1	step 2	step 3	
BM1 (s=1.75)	Acc. (%)	97.63	97.59	97.28	97.02	-0.61%
	Comm. (MB)	4.95	4.29	3.81	3.32	1.49× less
	Lat. (ms)	158	131	114	102	1.54× faster
BM2 (s=4)	Acc. (%)	98.64	98.44	98.37	98.13	-0.51%
	Comm. (MB)	38.28	28.63	24.33	15.76	2.42× less
	Lat. (ms)	158	144	134	104	1.51× faster
BM3 (s=2)	Acc. (%)	99.22	99.11	98.96	99.00	-0.22%
	Comm. (MB)	56.08	42.51	37.34	32.13	1.75× less
	Lat. (ms)	190	165	157	146	1.3× faster

Table 4.10: Trimming the BC2 network for CIFAR-10.

Property	Trimming Step				Change
	initial	step 1	step 2	step 3	
Acc. (%)	82.40	82.39	82.41	81.85	-0.55%
Com. (GB)	3.38	3.05	2.76	2.60	1.30× less
Lat. (s)	7.59	6.87	6.23	5.79	1.31× faster

4.7.5 Accuracy, Runtime, and Communication

Runtime and communication reports are available in Table 4.11 and Table 4.12 for MNIST and CIFAR-10 benchmarks, respectively. The corresponding neural network architectures are provided in Table A.1. Entries corresponding to a communication of more than 40GB are estimated using numerical runtime models.

Table 4.11: Accuracy (Acc.), communication (Comm.), and latency (Lat.) for MNIST dataset. Channel/neuron trimming is not applied.

Arch.	s	Acc. (%)	Comm. (MB)	Lat. (s)
BM1	1	97.10	2.57	0.12
	1.5	97.56	4.09	0.13
	2	97.82	5.87	0.13
	3	98.10	10.22	0.14
	4	98.34	15.62	0.15
BM2	1	97.25	2.90	0.10
	1.50	97.93	5.55	0.12
	2	98.28	10.09	0.14
	3	98.56	21.90	0.18
	4	98.64	38.30	0.23
BM3	1	98.54	17.59	0.17
	1.5	98.93	36.72	0.22
	2	99.13	62.77	0.3
	3	99.26	135.88	0.52
	4	99.35	236.78	0.81

Table 4.12: Accuracy (Acc.), communication (Comm.), and latency (Lat.) for CIFAR-10 dataset. Channel/neuron trimming is not applied.

Arch.	s	Acc. (%)	Comm. (MB)	Lat. (s)
BC1	1	0.72	1.26	3.96
	1.5	0.77	2.82	8.59
	2	0.80	4.98	15.07
	3	0.83	11.15	33.49
BC2	1	0.67	0.39	1.37
	1.5	0.73	0.86	2.78
	2	0.78	1.53	4.75
	3	0.82	3.40	10.35
BC3	1	0.77	1.35	4.23
	1.5	0.81	3.00	9.17
	2	0.83	5.32	16.09
	3	0.86	11.89	35.77
BC4	1	0.82	4.66	14.12
	1.5	0.85	10.41	31.33
	2	0.87	18.45	55.38
	3	0.88	41.37	123.94
BC5	1	0.81	5.54	16.78
	1.5	0.85	12.40	37.29
	2	0.86	21.98	65.94
	3	0.88	49.30	147.66
BC6	1	0.67	0.65	2.15
	1.5	0.74	1.46	4.55
	2	0.78	2.58	7.91
	3	0.80	5.77	17.44

4.8 Summary

The XONN framework was introduced in this chapter which is a novel approach to automatically train and use deep neural networks for the task of oblivious inference. XONN utilizes Yao's Garbled Circuits (GC) protocol and relies on binarizing the DL models in order to translate costly matrix multiplications to `XNOR` operations that are free in the GC protocol. Compared to Gazelle [JVC18], prior best solution, XONN achieves 7× lower latency. Moreover, in contrast to Gazelle that requires one round of interaction for each layer, our solution needs a

constant round of interactions regardless of the number of layers. Maintaining constant round complexity is an important requirement in Internet settings as a typical network latency can significantly degrade the performance of oblivious inference. Moreover, since our solution relies on the GC protocol, it can provide much stronger security guarantees such as security against malicious adversaries using standard cut-and-choose protocols. XONN high-level API enables clients to utilize the framework with a minimal number of lines of code. To further facilitate the adaptation of our framework, we design a compiler to translate the neural network description in Keras format to that of XONN.

4.9 Acknowledgements

This chapter, in part, has been published at the Proceedings of 2019 USENIX Security Symposium and appeared as M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, Farinaz Koushanfar, "XONN: XNOR-based Oblivious Deep Neural Network Inference" [RSC+19]. The dissertation author was the primary author of this material.

Chapter 5

HEAX: High-Performance Hardware Architecture for Computing on Encrypted Data

With the rapid increase in cloud computing, concerns surrounding data privacy, security, and confidentiality also have been increased significantly. Not only cloud providers are susceptible to internal and external hacks, but also in some scenarios, data owners cannot outsource the computation due to privacy laws such as GDPR, HIPAA, or CCPA. Fully Homomorphic Encryption (FHE) is a groundbreaking invention in cryptography that, unlike traditional cryptosystems, enables computation on encrypted data without ever decrypting it. However, the most critical obstacle in deploying FHE at large-scale is the enormous computation overhead.

In this chapter, we present HEAX, a novel hardware architecture for FHE that achieves unprecedented performance improvements. HEAX leverages multiple levels of parallelism, ranging from ciphertext-level to fine-grained modular arithmetic level. Our first contribution is a new highly-parallelizable architecture for number-theoretic transform (NTT) which can be of independent interest as NTT is frequently used in many lattice-based cryptography systems.

Building on top of NTT engine, we design a novel architecture for computation on homomorphically encrypted data. Our implementation on reconfigurable hardware demonstrates 164–268× performance improvement for a wide range of FHE parameters.

5.1 Introduction

Cloud computing has fundamentally changed the economics of computing in a short time. To mitigate security and privacy concerns of cloud computing, cloud providers should keep customers' data encrypted at all times. Symmetric-key encryption schemes, such as Advanced Encryption Standard (AES) [DR13], allow private data to be stored securely in a public cloud indefinitely. However, unless the customers share their secret keys with the cloud, the cloud becomes merely a storage provider.

In 2009, a new class of cryptosystems, called Fully Homomorphic Encryption (FHE) [Gen09], was introduced that allows arbitrary *computation on encrypted data*. This groundbreaking invention enables clients to encrypt data and send ciphertexts to a cloud that can evaluate functions on ciphertexts. Final and intermediate results are encrypted, and only the data owner who possesses the secret key can decrypt data, providing *end-to-end* encryption for the client.

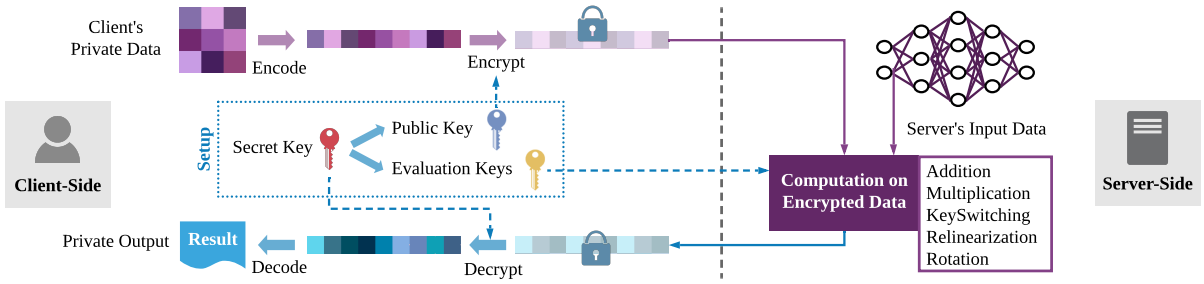


Figure 5.1: The data flow of an end-to-end encrypted computation based on homomorphic encryption.

Most FHE schemes, i.e., BGV [BGV12], BFV [FV12b], and TFHE [CGGI16] schemes, perform exact computation on encrypted data. A recently proposed FHE scheme called CKKS

[CKKS17] performs approximate computation of real numbers and supports efficient *truncation* of encrypted values. Several works [KSK+18, JKLS18b] have shown the benefits of choosing the CKKS scheme over other schemes when an approximate computation is required, e.g., in Machine Learning applications. Therefore, we focus on the CKKS scheme in this chapter, even though our core modules are applicable to most of the FHE schemes.

In this chapter, we introduce HEAX (stands for Homomorphic Encryption Acceleration): a novel high-performance architecture for computing on (homomorphically) encrypted data. We design several optimized core computation blocks for fast modular arithmetic and introduce a new architecture for high-throughput Number-Theoretic Transform (NTT). Building on top of the NTT module we design modules to perform high-level operations supported by FHE, thus accelerating any FHE-based privacy-preserving system.

Prior Art and Challenges. The *ciphertext* in FHE schemes is a set (usually a pair) of polynomials with degree $n-1$ (vectors of n integers) modulo a big integer. One of the main challenges of designing an architecture for FHE is that homomorphic operations on ciphertexts involve computationally intensive modular arithmetic on big integers (with several hundred bits). These operations have convoluted data dependency among different parts of the computation, making it challenging to design a high-throughput architecture. Moreover, the degree of the underlying polynomials is enormous (in the order of several thousand). Storing the entire intermediate results on FPGA chip is prohibitive.

Prior work that propose customized hardware for non-CKKS schemes have taken one of these approaches: (i) Designing co-processors that only accelerate certain low-level ring operations [CRS16, CGRS14, WH13, CMO+14, DÖS14b, JGCM+15]; high-level operations are performed on the CPU-side, which makes the co-processors of limited practical use. (ii) Storing intermediate results on off-chip memory, which significantly degrades the performance [PNPM15] to the extent that it can be worse than naive software execution [RJV+18]. (iii) Designing a hardware for a fixed modest-sized parameter, e.g., $n = 2^{12}$ [RTJ+19]. However, encryption

parameters determine the security-level and the maximum number of consecutive multiplications that one can perform on ciphertext, both of which are application-dependent. One of our primary design goals in HEAX is to have an architecture that can be readily used for a wide range of encryption parameters. In addition, we propose several techniques to efficiently store and access data from on-chip memory and minimize (or eliminate for some parameter sets) off-chip memory accesses.

Client-Side and Server-Side Computation. Figure 5.1 illustrates the data flow and the operations involved in a typical application based on FHE. The client encrypts her data and sends the resulting ciphertext to the cloud. The cloud server performs the computation on encrypted data and sends the result back to the client. In order to perform SIMD-style operations, an *encoding* step is performed by the client to embed many numbers in a single ciphertext. Note that encoding/decoding and encryption/decryption are performed on the client-side. These operations are not computationally expensive; thus, we do not implement customized hardware for these operations. The operations that are performed by the server for *evaluating* a function on ciphertexts are computationally intensive and are the focus of this work.

Contributions. In what follows, we elaborate on our major contributions in more detail:

- We design a novel architecture for number-theoretic transform which is a fundamental building block – and usually the computation bottleneck – for many lattice-based cryptosystems including all FHE schemes. Our design can process arbitrary-sized polynomials with an adjustable throughput. We develop several techniques to overcome the challenges due to the complex data-dependency and convoluted access patterns within NTT.

- We introduce the *first* architecture for CKKS homomorphic encryption. We leverage multi-layer parallelism design starting from ciphertext-level to fine-grained optimized modular arithmetic engines. In contrast to the prior art for other FHE schemes, our architecture can be scaled for different FPGA chips due to its modularity. Moreover, HEAX is not

custom-designed for specific FHE parameter set and can be used for a broad range of parameters.

- We provide a proof-of-concept implementation on two Intel FPGAs that represent two different classes of FPGAs in terms of available resources. We implement all high-level operations supported by CKKS and evaluate our design for three sets of FHE parameters. Our experimental results demonstrate more than *two orders of magnitude* performance improvement compared to heavily-optimized Microsoft SEAL library running on CPU.

5.2 MULT Module

In this section, we describe our proposed architectures for homomorphic multiplication.

5.2.1 Homomorphic Multiplication Algorithm

This operation is performed in RNS and NTT form. Although in general ciphertexts can have more than two polynomial components, in practice, ciphertexts are usually relinearized and the multiplication is carried out on two components as discussed next. Nevertheless, our proposed architecture is generic and supports any number of components.

• $\underline{\texttt{CKKS.Mul}(\texttt{ct}_0,\ \texttt{ct}_1)}$: Given ciphertexts $\texttt{ct}_0,\ \texttt{ct}_1 \in R_{q_\ell}^2$ encrypting $\texttt{pt}_0,\ \texttt{pt}_0 \in R$, generate $\texttt{ct}' \in R_{q_\ell}^3$ according to Algorithm 4 which encrypts $\texttt{pt}_0 \cdot \texttt{pt}_1 \in R$.

Algorithm 4 Homomorphic Mult. | $\texttt{CKKS.Mul}(\texttt{ct}_0,\ \texttt{ct}_1)$

Input: $\texttt{ct}_0 = (\tilde{\mathbf{A}}_0,\ \tilde{\mathbf{A}}_1),\ \texttt{ct}_1 = (\tilde{\mathbf{B}}_0,\ \tilde{\mathbf{B}}_1) \in (\prod_{i=0}^{\ell} R_{p_i})^2$
Output: $\texttt{ct} = (\tilde{\mathbf{C}}_0,\ \tilde{\mathbf{C}}_1,\ \tilde{\mathbf{C}}_2) \in (\prod_{i=0}^{\ell} R_{p_i})^3$

1: **for** $(i = 0;\ i \leq \ell;\ i = i+1)$ **do**
2: $\quad \tilde{\mathbf{c}}_{0,i} \leftarrow \texttt{Mod}(\tilde{\mathbf{a}}_{0,i} \odot \tilde{\mathbf{b}}_{0,i},\ p_i)$ ▷ Dyadic Core
3: $\quad \tilde{\mathbf{c}}_{1,i} \leftarrow \texttt{Mod}(\tilde{\mathbf{a}}_{0,i} \odot \tilde{\mathbf{b}}_{1,i} + \tilde{\mathbf{a}}_{1,i} \odot \tilde{\mathbf{b}}_{0,i},\ p_i)$
4: $\quad \tilde{\mathbf{c}}_{2,i} \leftarrow \texttt{Mod}(\tilde{\mathbf{a}}_{1,i} \odot \tilde{\mathbf{b}}_{1,i},\ p_i)$
5: **end for**

5.2.2 HEAX Word Size and Native Operations

Microsoft SEAL library [SEA19] is developed for x86 architectures with 64-bit native operations. However, on FPGAs, the bit-width of Digital Signal Processing (DSP) units that perform multiplication may vary, hence, it is more efficient to have a flexible bit-width for native operations. For example, the two FPGA chips that we have implemented our architecture on have 27-bit DSP units. Choosing 27-bit or 54-bit words enables us to use fewer DSPs to do the same computation. Naive construction of a 64-bit multiplier requires nine 27-bit DSPs. Whereas, a 54-bit multiplier requires only four. However, by reducing the bit-width of the RNS bases, one may need to increase the number of RNS bases; roughly speaking, by a factor of $\frac{64}{54} \approx 1.2$. In practice, small ciphertext moduli are usually less than 54 bits and thus, we do not need to increase the number of moduli.

It is worth-mentioning that leveraging more sophisticated multi-word multiplication algorithms such as Toom-Cook, one can implement 64-bit multiplication using *five* 27-bit multipliers together with more bit-level and Addition operations. Overall, by switching from 64-bit native operations to 54-bit, we observe between 1.4–2.25$\times$ *reduction in the number of DSP units* needed (depending on the HE parameters). However, to support 54-bit word size, we need to make sure that all of the ciphertext moduli (p_i) are (i) less than 52-bit to ensure the correctness of Algorithm 1 and (ii) congruent to 1 mod $2n$ to support NTT as described in Section 5.3. We have modified the SEAL library accordingly and precomputed all of such moduli for different parameter sets.

5.2.3 MULT Architecture

The `MULT` module can process both ciphertext-ciphertext (C-C) as well as ciphertext-plaintext (C-P) homomorphic multiplications. We describe the architecture for C-C multiplication as C-P is a special case of C-C. Since ciphertexts are in NTT form by default, homomorphic

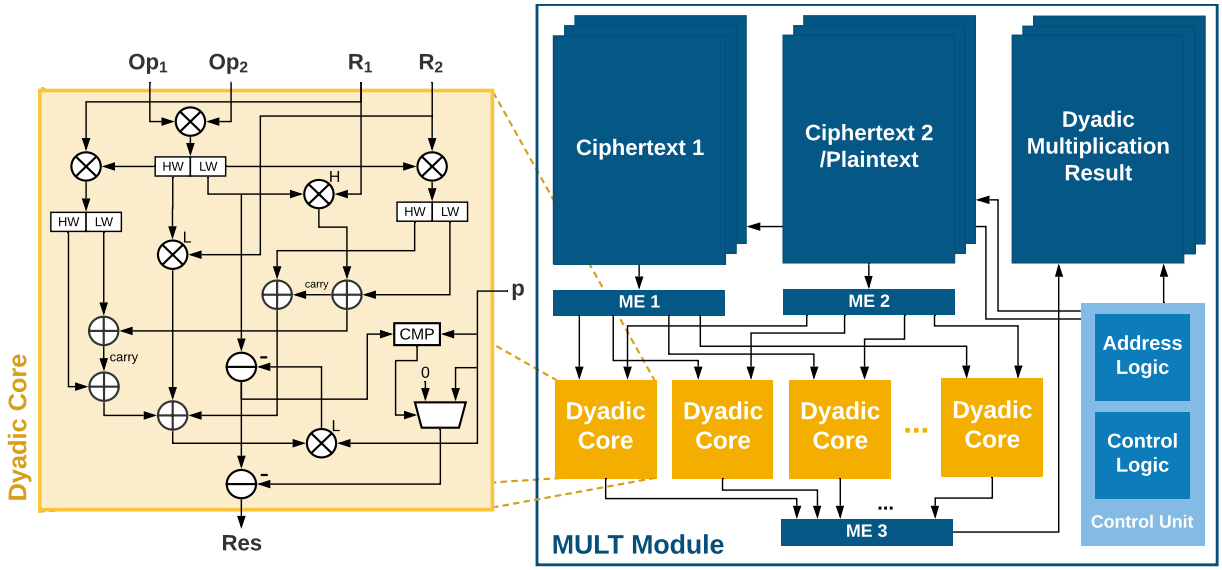


Figure 5.2: Architecture of MULT module.

multiplication is simply a series of *dyadic* products on different components.

The `MULT` module, as depicted in Figure 5.2, encompasses nc_{DYD}-many *Dyadic Cores*; thus, it can compute nc_{DYD} dyadic multiplication at each clock cycle (*nc* stands for number of cores). Each Dyadic core takes as input two polynomial coefficients (Op_1 and Op_2), two precomputed constant values (R_1 and R_2), and one-word prime p and outputs the result.

Let us denote the number of components in $\texttt{ct}_0$ and $\texttt{ct}_1$ by α and β, respectively. The outcome of homomorphic multiplication is a ciphertext with $\alpha+\beta-1$ components. Each ciphertext component is represented in a RNS form. Recall that in homomorphic multiplication (Algorithm 4), the computation can be carried out independently on each RNS basis. We leverage this property to reduce BRAM utilization. Minimum BRAM utilization is achieved by storing only one residue of one ciphertext component on FPGA chip. However, this approach significantly increases data transfer from CPU to FPGA from $(\alpha+\beta)\cdot n$ words to $(\alpha\cdot\beta+\min(\alpha,\beta))\cdot n$ words because we need to compute all pairwise combinations of $\texttt{ct}_0$ and $\texttt{ct}_1$ components. Thus, we allocate α-many memories of size n for $\texttt{ct}_0$ and β-many memories for $\texttt{ct}_1$ to hold *one* residue of *all* ciphertext components. As a result, we achieve $O\big((\alpha+\beta)\cdot n\big)$ data transfer and BRAM consumption.

In order to fully utilize all nc_{DYD} Dyadic cores – regardless of the values of α and β – we

read nc_{DYD} coefficients from one of the polynomials of $\texttt{ct}_0$ and $\texttt{ct}_1$ at every clock cycle. However, each unit of on-chip memory, i.e., Block RAMs (BRAM), only supports one read and one write at each clock cycle. In order to read many coefficients from one polynomial at each cycle, we store each polynomial across nc_{DYD}-many parallel memory blocks that share common read/write address signals as depicted in Figure 5.2. Let us call the aggregation of one row among different BRAMs as a *memory element* (ME). Therefore, at every cycle, one memory element (ME1/ME2) is read from $\texttt{ct}_0$/$\texttt{ct}_1$ memory banks and the result (ME3) is written to a separate output memory.

5.3 NTT Module

NTT calculation as well as its inverse (INTT) are the most computationally intensive *low-level* operations. Polynomial multiplication is more efficiently performed by transforming polynomials and using the convolution theorem. In what follows, we provide an overview on NTT algorithm followed by our proposed architecture.

5.3.1 Algorithms

Computing $\mathbf{c} = \mathbf{a} \cdot \mathbf{b} \in R_p$ is equivalent to computing the negacyclic convolution of their coefficient vectors in $\mathbb{Z}_p^n$: $c_j = \sum_{i=0}^{j} a_i b_{j-i} - \sum_{i-j+1}^{n-1} a_i b_{j-i+n} \pmod{p}$, $j = 0,\ 1,\ \ldots,\ n-1$. For a large n it is asymptotically better to use the convolution theorem and perform a specific form of fast Fourier transform, i.e., NTT, over a finite field. Polynomials are kept in NTT form to reduce the number of NTT/INTT conversions. Fast NTT algorithms are well studied in lattice-based cryptography. We adapt the algorithms in [LN16] which analyzes fast NTT algorithms and introduces specific optimizations for negacyclic convolution. For a ring degree n, we choose a prime number $p = 1 \bmod 2n$ such that there exists a $2n$-th primitive root of unity ψ, i.e., $\psi^n = -1 \bmod p$.

• $\underline{\texttt{NTT}_p(\mathbf{a})}$: Given $\mathbf{a} \in \mathbb{Z}_p^n$, compute $\tilde{\mathbf{a}} \in \mathbb{Z}_p^n$ such that $\tilde{a}_j = \sum_{i=0}^{n-1} a_i \psi^{(2i+1)j}$, according to Algo-

rithm 5.

An important operation that is used during key switching and rescaling is called *flooring* which can be formalized as: • $\underline{\texttt{Floor}(\tilde{\mathbf{C}}, p)}$: Given $\tilde{\mathbf{C}}$, the RNS and NTT form of $\mathbf{c} \in R_{q_\ell p}$, generate $\tilde{\mathbf{C}}'$, the RNS and NTT form of $\mathbf{c}' = \lfloor p^{-1} \cdot \mathbf{c} \rceil \in R_{q_\ell}$ according to Algorithm 6.

5.3.2 NTT Architecture

In what follows, we use the term NTT to refer to both NTT and INTT operations/modules for simplicity. At the end of this section, we discuss the differences between these two modules. As can be seen from Algorithm 7, in `KeySwitch`, NTT is frequently used in different parts of this algorithm. However, the number of required transformations is not consistent in different parts of the Algorithm. In order to have a fully-pipelined architecture, we allocate one NTT module per each NTT operation in Algorithm 7. However, the relative throughput-rate among different NTT instances depends on the chosen FHE parameters, which is application-dependent. As a result, we need to have a generic architecture such that the *throughput can be adjusted* as needed. This, in turn, is translated to the number of NTT *cores* that is dedicated to a given NTT module.

NTT Core. Figure 5.3 shows the internal architecture of an NTT core. Each core accepts two coefficients ($c_{in.a}$ and $c_{in.b}$), one twiddle factor (w), one precomputed value (wp), and a prime number (p) as inputs and computes two transformed coefficients as the outputs ($c_{out.a}$ and $c_{out.b}$). The modular arithmetic operations within NTT core are all pipelined to maximize the throughput of the overall NTT module.

Figure 5.3 illustrates the full architecture of NTT module. From the functionality perspective, the architecture follows Algorithm 5. At a high-level, the NTT module computes NTT of a polynomial of size n in $\log n$ *stages*. In each stage, the module computes the transformed result of $2nc_{\text{NTT}}$ coefficients, thus, requiring $\frac{n}{2nc_{\text{NTT}}}$ *steps* to finish one stage.

On the three corners of the NTT architecture exist data memory, twiddle factor memories, and the output memory. At every cycle, one ME is fetched from data memory and is stored in

Algorithm 5 Number-Theoretic Transform (NTT) | $\texttt{NTT}_p(\mathbf{a})$

Input: $\mathbf{a} \in \mathbb{Z}_p^n$, $p \equiv 1 \mod 2n$, $\mathbf{Y} \in \mathbb{Z}_p^n$ storing powers of ψ in bit-reverse order, and $\mathbf{Y}' = \lfloor \mathbf{Y} \cdot 2^w / p \rfloor$.
Output: : $\tilde{\mathbf{a}} \leftarrow \texttt{NTT}_p(\mathbf{a})$ in bit-reverse ordering.

for $(m = 1;\ m < n;\ m = 2m)$ **do**
  for $(i = 0;\ i < m;\ i++)$ **do**
    for $(j = \frac{i \cdot n}{m};\ j < \frac{(2i+1)n}{2m};\ j++)$ **do**
      $v = \texttt{MulRed}(a_{j+\frac{n}{m}},\ y_{m+i},\ y'_{m+i},\ p)$
      $a_{j+\frac{n}{m}} = a_j - v \pmod{p}$
      $a_j = a_j + v \pmod{p}$ ▷ NTT Core
    end for
  end for
end for
$\tilde{\mathbf{a}} \leftarrow \mathbf{a}$

Algorithm 6 RNS Flooring | $\texttt{Floor}(\tilde{\mathbf{C}}, p)$

Input: $\tilde{\mathbf{C}} = (\tilde{\mathbf{c}}_0,\ \ldots,\ \tilde{\mathbf{c}}_{\ell+1}) \in \mathbb{Z}_{p_0}^n \times \cdots \times \mathbb{Z}_{p_\ell}^n \times \mathbb{Z}_p^n$.
Output: $\tilde{\mathbf{C}}' = (\tilde{\mathbf{c}}'_0,\ \ldots,\ \tilde{\mathbf{c}}'_\ell) \in \mathbb{Z}_{p_0}^n \times \cdots \times \mathbb{Z}_{p_\ell}^n$.

$\mathbf{a} \leftarrow \texttt{INTT}_p(\tilde{\mathbf{c}}_{\ell+1})$ ▷ INTT Module
for $(i = 0;\ i \leq \ell;\ i = i + 1)$ **do**
  $\overline{\mathbf{r}} \leftarrow \texttt{Mod}(\mathbf{a},\ p_i)$
  $\tilde{\mathbf{r}} \leftarrow \texttt{NTT}_{p_i}(\overline{\mathbf{r}})$ ▷ NTT Module
  $\tilde{\mathbf{c}}'_i \leftarrow \tilde{\mathbf{c}}_i - \tilde{\mathbf{r}} \pmod{p_i}$
  $\tilde{\mathbf{c}}'_i \leftarrow \texttt{Mod}\left(\left[p^{-1}\right]_{p_i} \cdot \tilde{\mathbf{c}}'_i,\ p_i\right)$ ▷ MS Module
end for

ME_e and ME_o registers every other cycles, respectively. For each input coefficient of NTT cores, i.e., $\text{c}^{\ell}_{\text{in.a}}$ or $\text{c}^{\ell}_{\text{in.b}}$, a set of multiplexers select the correct coefficient from ME_e and ME_o (depicted as light blue boxes in Figure 5.3).

The throughput is proportional to the number of NTT cores. We denote the number of NTT cores as nc_{NTT}. Ideally at each clock cycle, and given full utilization of NTT cores, $2nc_{\text{NTT}}$ coefficients are transformed. In order to read and write $2nc_{\text{NTT}}$ coefficients at each clock cycle, we store each polynomial across many parallel BRAMs that share common read/write address signals as depicted in Figure 5.3 (similar to the `MULT` module). This is possible thanks to the *aligned* access pattern in NTT: while access pattern changes during NTT, the number of consecutive accesses to the polynomial is always a power of two. Next, we discuss the details of the access

patterns in NTT followed by our proposed solution to select each coefficient efficiently.

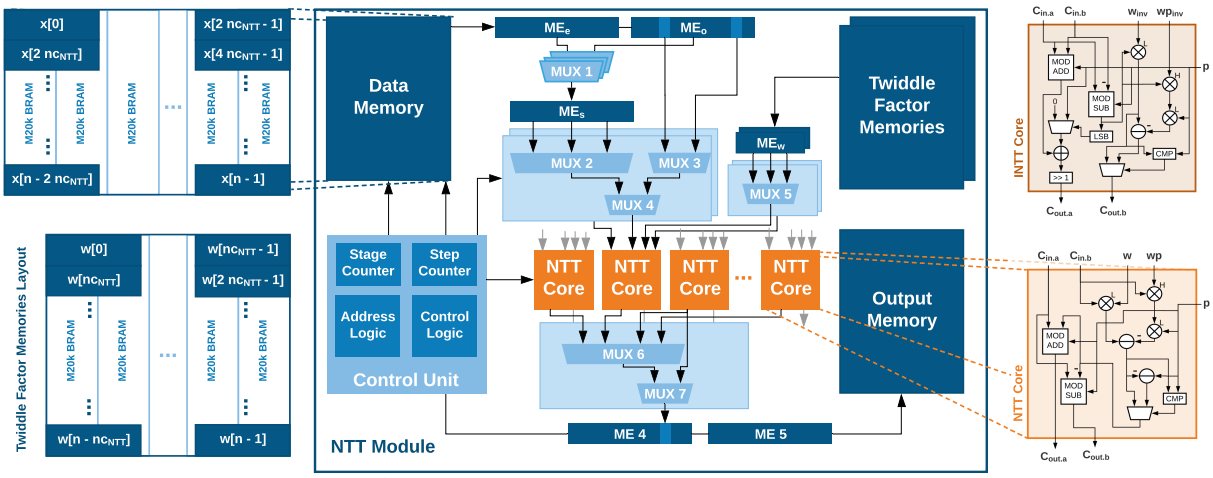


Figure 5.3: Architecture of NTT module.

5.3.3 Access Pattern

One of the main challenges in realizing the proposed NTT architecture is that the access pattern of the coefficients changes from one stage to another. We categorize the access patterns into two groups as illustrated in Figure 5.4. During the first $(\log n - \log nc_{\text{NTT}} - 1)$ stages, each pair of coefficients for each NTT core are stored in different MEs. Let us call these *Type 1* stages. For instance, consider $n = 4096$ and $nc_{\text{NTT}} = 8$, during the first step of the first stage of NTT, $x[0]$ (in ME_0) and $x[2048]$ (in ME_{256}) should be passed to the first NTT core. More precisely, polynomial coefficient $x[j]$ $(j = 0, 1, \ldots, \frac{n}{2} - 1)$ is passed together with $x[j + \frac{n}{2}]$ to a given NTT core. In general, during i^{th} stage, $x[j+m]$ $(j = 0, 1, \ldots, \frac{n}{2^{1+i}} - 1)$ is passed along with $x[j + m + \frac{n}{2^{1+i}}]$ where $m \in \{\frac{h \cdot n}{2^i} \mid h = 0, 1, ..., i\}$. The address of the ME that is fetched in Type 1 stages is computed in *Address Logic*. As soon as $\frac{n}{2^i} = 2nc_{\text{NTT}}$, the inter-ME data dependency no longer exists, and pairs of coefficients are selected from within each ME independently, i.e., *Type 2* stages.

In Type 1 stages, coefficients within two fetched MEs are always accessed in the same order. For example, the second coefficient in each ME is always passed to the second NTT core.

However, in Type 2 stages, a coefficient at specific location of ME is passed to a different NTT core or even different inputs of an NTT core. Therefore, coefficients have to be *reordered* to be passed to NTT cores. Later in this section, we discuss an efficient method for this task.

The access pattern for twiddle factors, i.e., $\mathbf{Y}$ and $\mathbf{Y}'$ in Algorithm 5, is different. At stage i, only 2^i *unique* values of twiddle factors, starting at index 2^i of twiddle polynomial, are used. Since in the worst-case scenario, nc_{NTT} unique twiddle factors are used in a single *step* of NTT, we store twiddle factors in batches of size nc_{NTT} in parallel.

5.3.4 Reordering Coefficients and Optimal MUXs

During Type 1 stages, once the ME is fetched, passing each coefficient within ME to the right NTT core (and right input wire) is straightforward and it can be summarized as follows:

$$\begin{cases} \text{c}^{\ell}_{\text{in.a}} = \text{ME}_e[\ell + (j \bmod 2) \cdot nc_{\text{NTT}}] \\ \text{c}^{\ell}_{\text{in.b}} = \text{ME}_o[\ell + (j \bmod 2) \cdot nc_{\text{NTT}}] \end{cases}$$

where $\text{c}^{\ell}_{\text{in.a}}$ (respectively $\text{c}^{\ell}_{\text{in.b}}$) is the input coefficient a (respectively b) of ℓ^{th} NTT core, ME_e (resp. ME_o) is the memory element at "even" ("odd") read cycles, i.e., $j \bmod 2 = 0$ ($j \bmod 2 = 1$) where j is the step number. In other words, $\text{c}^{\ell}_{\text{in.a}}$ (resp. $\text{c}^{\ell}_{\text{in.b}}$) is selected from one of the two positions from ME_e (resp. ME_o) using multiplexer #3 (`MUX3`).

In Type 2 stages, first one of the ME_e or ME_o is selected using $2nc_{\text{NTT}}$-many two-to-one multiplexers (`MUX1`) and is stored in ME_s registers. Next, $\text{c}^{\ell}_{\text{in.a}}$ (or $\text{c}^{\ell}_{\text{in.b}}$) receive data from one of the coefficients in ME_s depending on the value of ℓ and i. The naive approach is to use one multiplexer per each coefficient input of every NTT core that selects one number from $2nc_{\text{NTT}}$ fetched numbers. We denote such multiplexer as $\text{MUX}^{2nc_{\text{NTT}}}$. As a result, we need $2nc_{\text{NTT}}$-many $\text{MUX}^{2nc_{\text{NTT}}}$ to pass coefficients to NTT cores and the same number of MUXs to reorder them to be written back to the memory.

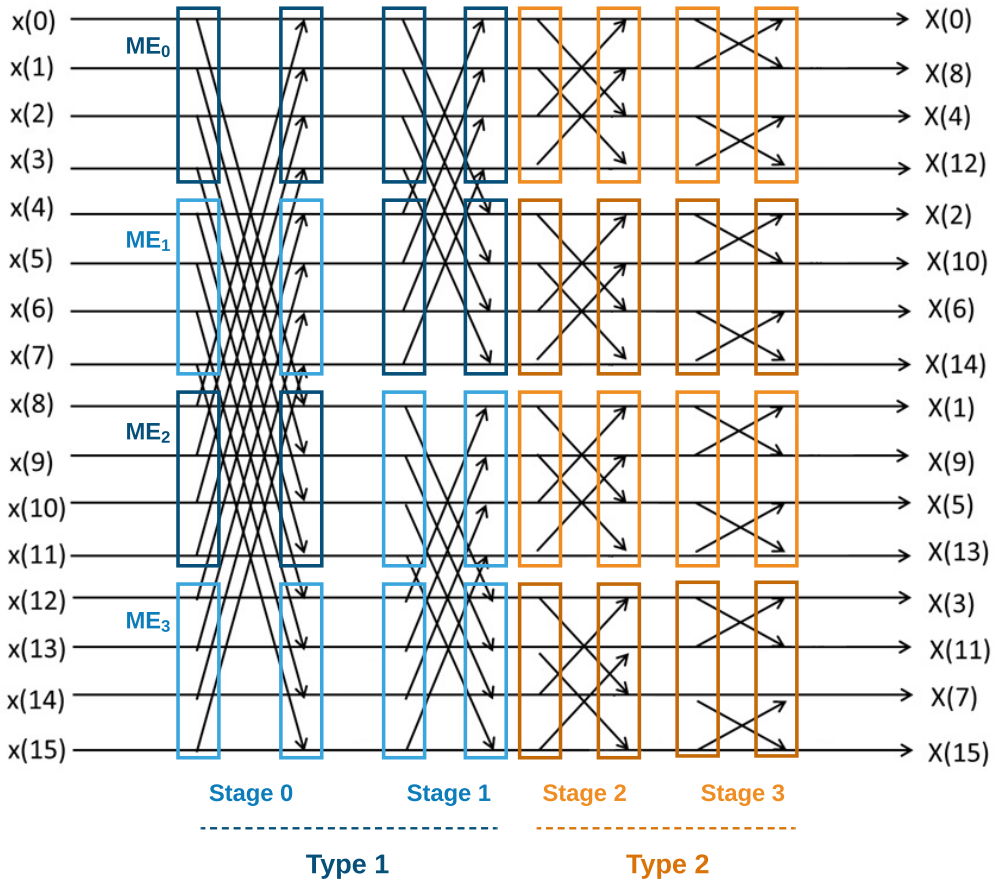


Figure 5.4: Access pattern of Type 1 and Type 2 stages in NTT.

These MUXs not only make the placement and route process more challenging but also consume enormous number of registers and logic blocks. Moreover, scaling the NTT module to higher number of cores (> 32) is inefficient due to super-linear resource consumption with respect to nc_{NTT}. In our case, synthesis tools failed to place and route the required resources to realize these MUXs. In contrast, we take advantage of the observation that *NTT cores' inputs have a different number of possibilities from which they select the correct coefficient at a given stage*. For example, during Type 2 stages, $c^0_{\text{in.a}}$ only receives coefficients from the first word of the fetched ME, regardless of the stage or step number.

In the worst-case scenario, there are $\log nc_{\text{NTT}}$ different indices from which a coefficient should be accessed from ME_s for a particular NTT core input. Therefore, instead of using $(4 \cdot nc_{\text{NTT}})$-many $\text{MUX}^{2nc_{\text{NTT}}}$, we instantiate $(4 \cdot nc_{\text{NTT}})$-many MUXs of size at most $\text{MUX}^{\log 2nc_{\text{NTT}}}$.

These optimal multiplexers are shows as `MUX2` in Figure 5.3. The selection signal of these MUXs is set to $s = \log n - 1 - i$ (i being the stage number). The corresponding inputs ($\text{MUX}\{c^{\ell}_{\text{in.a}}\}(\alpha)$ and $\text{MUX}\{c^{\ell}_{\text{in.b}}\}(\alpha)$) from which a coefficient should be selected are assigned based on the following formula:

$$= \begin{cases} \text{ME}_s[(\ell \,\&\, (2^s - 1)) + ((i >> s) << (s + \ell))] \\ \text{ME}_s[(\ell \,\&\, (2^s - 1)) + ((i >> s) << (s + \ell)) + 2^s] \end{cases}$$

where $\text{MUX}\{c^{\ell}_{\text{in.a}}\}(\alpha)$ is the α-th input wire of the MUX that selects the corresponding input coefficient from ME_s for the ℓ-th core, thus, $0 \leq \alpha < \log nc_{\text{NTT}}$. Finally, depending on the stage type, `MUX4` selects the output of `MUX2` or `MUX3`.

A similar set of MUXs (`MUX6` and `MUX7`) are used to reorder the data back before storage. Final results (`ME4` and `ME5`) will be stored in the data memory during two consecutive clock cycles; except for the the last stage where they will be stored in *output memory*. The optimized multiplexers for twiddle factors is designed in a similar manner. The optimal multiplexers are an integral part of the design of NTT module. For instance, this optimization reduces the number of registers used for a 8-core NTT module from 224,000 to 97,000. Note that to synthesize the NTT module, register is the most limited resource (see Section 5.6), thus, without the proposed optimal multiplexers, one cannot scale the design properly.

5.3.5 NTT High-Level Pipeline

Storing polynomial coefficients across parallel memory blocks enables simultaneous access to multiple coefficients. However, the NTT cores cannot be fully utilized due to the following reason. During Type 1 stages, coefficients that should be passed to each NTT core are not located in the same ME. Therefore, two different MEs should be read before the computation can start which introduces 50% bubble in the NTT core pipeline. More precisely, first $\log n -$

$\log nc_{\text{NTT}} - 1$ stages face this problem. Given that NTT modules consume most of the FPGA resources, this issue reduces the throughput of the entire design to $(\log n - \log nc_{\text{NTT}} - 1)/\log n$.

To address this problem, we propose to double the size of MEs and store $2nc_{\text{NTT}}$ consecutive coefficients in each memory element. Meanwhile, we reduce the depth of the memories that store the polynomial by half. Even though it is still necessary to read two MEs before starting the computation, we can now transform *two* MEs in the next two cycles and store them back in the memory. This modification results in the full utilization of NTT core. In order to have minimal BRAM usage, all of the reads and writes during different NTT stages are *inplace*, and no additional BRAM is used to store intermediate values.

5.3.6 Memory Utilization and Word-Packing

Storing multiple polynomial coefficients in multiple parallel memory units (M20K) causes memory block under-utilization both depth-size and width-size. Consider a general scenario where β-many numbers are stored in parallel:

Depth-wise: Each M20K memory unit holds 512-many 40-bit wide words and at any cycle, one word can be read from or written into the memory. When fewer than 512 words are stored in the memory, the rest of the memory rows cannot be used to store a secondary polynomial since at any point in time we are reading/writing one word associated with the first polynomial. As long as $\frac{n}{\beta} \geq 512$, M20K is fully utilized. This inequality generally holds in our architecture except when $n = 2^{12}$ (smallest polynomial size) and $nc_{\text{NTT}} = 16$ which makes M20K *half* utilized. However, this is not an issue since our design is not BRAM-constrained when $n = 2^{12}$.

Width-wise: As the polynomial-size (n) grows, our design becomes more and more BRAM-constrained to the extent that at $n = 2^{14}$, there is not enough BRAM on the chip; thus, we have to use DRAM as well (we will discuss this in more detail in Section 5.5). Therefore, it is essential that the polynomials are efficiently stored in memory. By storing each coefficient in a separate physical BRAM, we will only reach $\frac{54}{2\cdot 40} = 68\%$ utilization. In contrast, we pack

multiple coefficients and store them in fewer M20K units as shown in Figure 5.3 reaching memory utilization of $\beta \cdot 54/(\lceil \beta \cdot 54/40 \rceil \cdot 40)$. For $\beta = 8$, BRAM utilization will reach more than 98%.

Performance. Computing the NTT of a polynomial requires $\log n$ stages and each stage takes $\frac{n}{2nc_{\text{NTT}}}$ cycles. Hence, it takes $\frac{n \log n}{2nc_{\text{NTT}}}$ cycles to compute one NTT.

INTT Module. This module is identical to the NTT module except: (i) the NTT core is replaced by the INTT core, (ii) the control unit operates in the reverse order of stage numbers, and (iii) twiddle factors correspond to the INTT calculations.

5.4 KeySwitch Module

In this section, we discuss the `KeySwitch` algorithm followed by our proposed architecture and the design details.

5.4.1 Algorithm

Key switching is a technique to make a ciphertext decryptable with a different secret key homomorphically. Various gadget decomposition methods can be adopted to balance noise growth and execution time. Given q_{d-1}, the product of coprime integers $p_0, \ \ldots, \ p_{d-1}$, and q_ℓ divides q_{d-1}, define gadget decomposition $R_{q_\ell} \mapsto R^d$ as $\mathbf{g}^{-1}(\mathbf{a}) = \left([\mathbf{a}]_{p_i} \right)_{0 \le i \le d-1}$, and gadget vector as $\mathbf{g} = \left(\pi_i \left[\pi_i^{-1} \right]_{p_i} \right)_{0 \le i \le d-1}$ where $\pi_i = \frac{q_{d-1}}{p_i}$. This choice of gadget decomposition contributes to a fast key switching and high noise growth. With the special modulus p and a rescaling at the end of key switching, explained in [CDKS19], key switching is almost noise-free.

• `KeySwitch(ct, ksk)`: Given a ciphertext $\texttt{ct} = (\mathbf{c}_0, \mathbf{c}_1) \in R_{q_\ell}^2$ decryptable with secret key $\mathbf{s}$ and a key switching key $\texttt{ksk} = (\mathbf{D}_0 \mid \mathbf{D}_1) \in R_{q_\ell p}^{(L+2)\times 2}$, where $\mid$ appends one column vector to another, generate a new ciphertext $\texttt{ct}' = (\mathbf{c}_0', \ \mathbf{c}_1') \in R_{q_\ell}^2$ decryptable with secret key $\mathbf{s}'$. (see Algorithm 7).

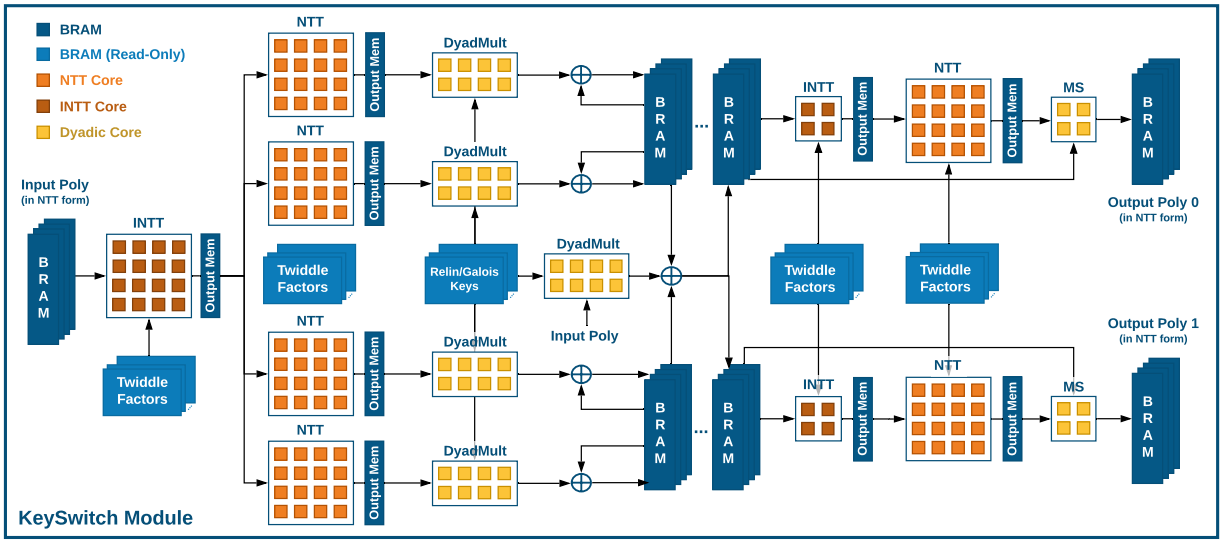


Figure 5.5: Architecture of KeySwitch module.

5.4.2 KeySwitch Architecture

`KeySwitch` is the most computationally intensive *high-level* operation in CKKS. It has several important roles, including relinearization and ciphertext rotation. Figure 5.5 illustrates the `KeySwitch` architecture, which from the functionality perspective corresponds to Algorithm 7. To reduce on-chip memory usage, our design takes one polynomial (one RNS component) at a time and outputs two polynomials. Recall that in CKKS, all polynomials are in NTT form by default. Thus, once the input polynomial is written into the input memory, it has to be converted back to the original domain. This process is performed using the first INTT module (`INTT0`). Next, the polynomial is transformed to the NTT form for all other primes (including the special modulus).

Since per each INTT computation, we have to perform k NTT, the throughput of the NTT module(s) has to be k-times the throughput of `INTT0`. Here, k is the number of RNS components of ciphertext modulus, i.e., $L+1$. This requirement can be realized in two different ways: (i) having one NTT module with k-many more cores than `INTT0` or (ii) having multiple NTT module with fewer cores per each module. We denote this NTT module (or a set of them) as `NTT0`. We will discuss the trade-offs later in this section. In Figure 5.5, the second approach (using more than one NTT module) is chosen for $n = 2^{13}$ and $k = 4$ parameter set.

Algorithm 7 Key Switching | `KeySwitch(ct, ksk)`

Input: $\texttt{ct} = (\tilde{\mathbf{C}}_0,\ \tilde{\mathbf{C}}_1) \in (\prod_{i=0}^{\ell} R_{p_i})^2$, and $\texttt{ksk} = \left(\left(\tilde{\mathbf{D}}_{i,0}\right)_{0\le i\le L+1} \mid \left(\tilde{\mathbf{D}}_{i,1}\right)_{0\le i\le L+1} \right) \in (p\prod_{i=0}^{L} R_{p_i})^{(L+2)\times 2}$

Output: $\texttt{ct}' = (\tilde{\mathbf{C}}'_0,\ \tilde{\mathbf{C}}'_1) \in (\prod_{i=0}^{\ell} R_{p_i})^2$

for $(i = 0;\ i \le \ell;\ i = i+1)$ **do**
  $\overline{\mathbf{a}} \leftarrow \texttt{INTT}_{p_i}(\tilde{\mathbf{c}}_{1,i})$ ▷ INTT Module
  for $(j = 0;\ j \le \ell;\ j = j+1)$ **do**
    if $i \neq j$ **then**
      $\overline{\mathbf{b}} \leftarrow \texttt{Mod}(\overline{\mathbf{a}},\ p_j)$
      $\tilde{\mathbf{b}} \leftarrow \texttt{NTT}_{p_j}(\overline{\mathbf{b}})$ ▷ NTT Module
    else
      $\tilde{\mathbf{b}} \leftarrow \tilde{\mathbf{a}}$
    end if
    $\tilde{\mathbf{c}}''_{0,j} \leftarrow \tilde{\mathbf{c}}''_{0,j} + \tilde{\mathbf{b}} \odot \tilde{\mathbf{d}}_{i,0,j} \pmod{p_j}$
    $\tilde{\mathbf{c}}''_{1,j} \leftarrow \tilde{\mathbf{c}}''_{1,j} + \tilde{\mathbf{b}} \odot \tilde{\mathbf{d}}_{i,1,j} \pmod{p_j}$ ▷ Dyadic Mod.
  end for
  $\overline{\mathbf{b}} \leftarrow \texttt{Mod}(\overline{\mathbf{a}},\ p)$
  $\tilde{\mathbf{b}} \leftarrow \texttt{NTT}_{p}(\overline{\mathbf{b}})$ ▷ NTT Module
  $\tilde{\mathbf{c}}''_{0,\ell+1} \leftarrow \tilde{\mathbf{c}}''_{0,\ell+1} + \tilde{\mathbf{b}} \odot \tilde{\mathbf{d}}_{0,i,L+1} \pmod{p_j}$
  $\tilde{\mathbf{c}}''_{1,\ell+1} \leftarrow \tilde{\mathbf{c}}''_{1,\ell+1} + \tilde{\mathbf{b}} \odot \tilde{\mathbf{d}}_{1,i,L+1} \pmod{p_j}$ ▷ Dyd. M.
end for
$\texttt{ct}' \leftarrow (\texttt{Floor}(\tilde{\mathbf{C}}''_0, p), \texttt{Floor}(\tilde{\mathbf{C}}''_1, p))$ ▷ INTT/NTT/MS
$\texttt{ct}' \leftarrow \texttt{CKKS.Add}(\texttt{ct}, \texttt{ct}')$

Once the NTT computations are finished, the `DyadMult` module computes the dyadic product between the output of NTT modules and the relinearization/Galois keys according to Algorithm 7. Recall that a dyadic product on the original input polynomial is also needed in `KeySwitch`; thus, a separate Dyadic module is used. After dyadic product computation, the result is stored in the corresponding memory banks. There are two *sets* of BRAM banks, each bank containing the RNS components of one polynomial.

The computation flow described above repeats for k-many times (one per each RNS component). The result is accumulated in the BRAM banks. After k iterations, the second part of the computation – usually referred to as *Modulus Switching* (developed in [BV11]) – is performed. In Modulus Switching which executes `Floor`, the polynomial corresponding to the *special modulus* has to be transformed back to the time domain (by `INTT1`) and then be

transformed using every other k primes (by `NTT1`). The aforementioned process is independently performed for both sets of banks. Next, the polynomial is multiplied by the inverse value of the associated prime and subtracted from the result of the first half of `KeySwitch` computation. The `MS` module embeds multiplication and subtraction operations. The output of `KeySwitch` is stored as two sets of k polynomials referred to as "Output Poly 0/1".

5.4.3 Balancing Throughput

Our primary goal in designing `KeySwitch` architecture is to have a fully *end-to-end pipelined* module that can process many key switching operations simultaneously without excessive FIFOs between different components. Thus, we have to tune the throughput of each component carefully. As we discussed in Section 5.3, this is one of the reasons to design a flexible architecture for NTT, the throughput of which can be adjusted. According to Algorithm 7, per each initial INTT, we have to compute k NTTs. In general, let's denote the number of `NTT0` as m_0 (assuming a power of two number). Thus, we have: $nc_{\text{NTT}_0} = k \cdot nc_{\text{INTT}_0}/m_0$.

Next, we compute the number of cores needed for `DyadMult`. Recall that it takes $(n \ \log n)/(2nc_{\text{NTT}_0})$ cycles for NTT module to finish the computation. The `DyadMult` module has to compute the product of NTT output with two different sets of keys ($\texttt{ksk} = \mathbf{D}_0 \mid \mathbf{D}_1$). It takes $(2n)/nc_{\text{DYD}}$ cycles to perform dyadic multiplication on the output of the NTT module. Since in general, $\log n$ is not a power of two, the throughputs do not perfectly match. We make sure that the throughput of Dyadic module is greater than that of (or equal to) the NTT module by satisfying the following inequality:

$$\frac{2n}{nc_{\text{DYD}}} \leqslant \frac{n \ \log n}{2nc_{\text{NTT}_0}} \Rightarrow nc_{\text{DYD}} = \left\lceil \frac{4nc_{\text{NTT}_0}}{\log n} \right\rceil$$

The throughput of `INTT1` modules can be adjusted by assigning $nc_{\text{INTT}_1} = \lceil nc_{\text{INTT}_0}/k \rceil$. One can also determine $nc_{\text{NTT}_1} = nc_{\text{INTT}_0}$ and $nc_{\text{MS}} = \lceil (2nc_{\text{NTT}_1})/\log n \rceil$. For two FPGA chips that we

have implemented HEAX on, the optimal architecture parameters are computed and summarized in Table 5.5.

5.4.4 KeySwitch Ops. and Synchronization

Figure 5.6 shows the high-level pipeline of `KeySwitch` module for $n = 2^{13}$ (third row of Table 5.5). All of the modules – and their internal components – are pipelined, and the throughput is balanced. As can be seen, multiple key switching operations are computed simultaneously in different pipeline stages (in lighter colors). The fifth Dyadic module that operates on input polynomial BRAM is *synchronized* with the rest of the Dyadic modules even though the computation can be started as soon as the input poly is available. The reason is that during each of the k iterations of Dyadic product, each module computes and accumulates the results by reading/writing from/to a separate BRAM bank. This enables us to avoid any *memory replication* considering that these memory banks are prohibitively large. However, this *delayed* computation introduces a dependency problem in the pipeline referred to as "Data Dependency 1". By the time the k-th Dyadic module starts the computation, the content of input poly is overridden by the next key switching operation. As a result, we allocate enough BRAM to hold f_1-many polynomials where $f_1 = \left\lceil 3 + \frac{nc_{\text{INTT}_0}}{nc_{\text{NTT}_0}} \right\rceil$. Similarly, `MS` module receives inputs from `DyadMult` modules. This is marked as "Data Dependency 2" in Figure 5.6. Thus, we need to allocate more memory to store the output of the `DyadMult` modules in f_2 different buffers. The value of f_2 can be computed as: $f_2 = \left\lceil 1 + m_0 \cdot \frac{nc_{\text{INTT}_1}}{nc_{\text{NTT}_1}} + \frac{nc_{\text{INTT}_1} \cdot \log n}{nc_{\text{MS}}} \right\rceil$.

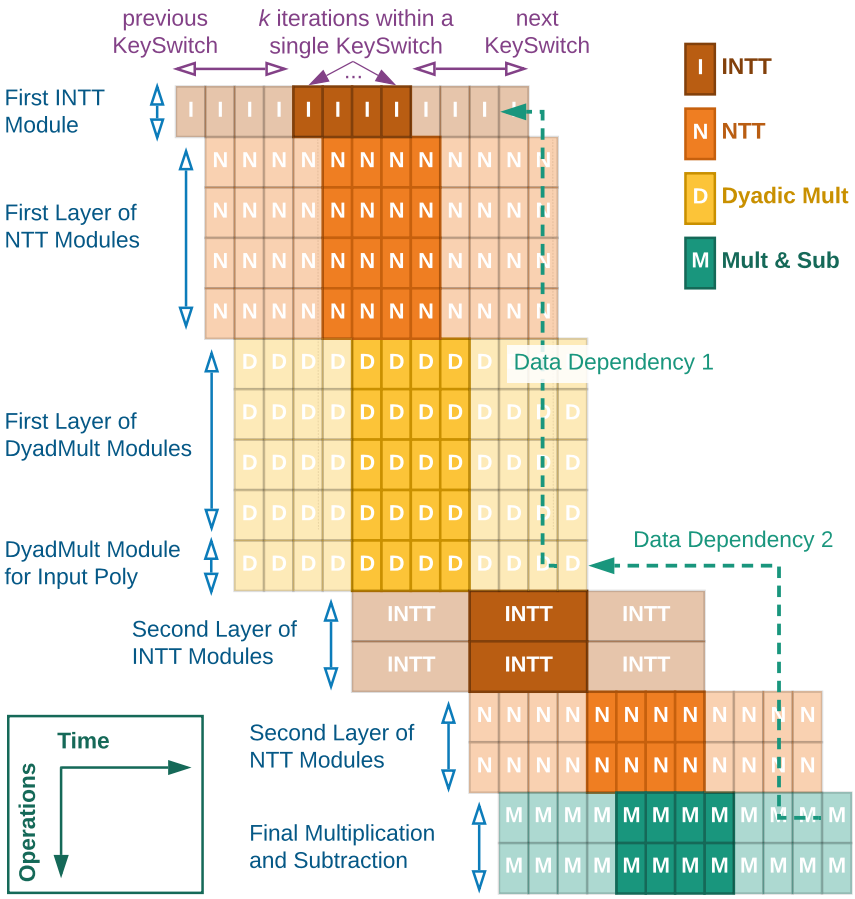


Figure 5.6: High-level pipeline of KeySwitch module.

5.5 System View and Data Flow

In this section, we discuss a higher-level view of the computation and elaborate on the data flow. Figure 5.7 shows a system-view comprising host CPU and FPGA Board which are connected via Peripheral Component Interconnect express (PCIe) bus. On FPGA board, exist the FPGA chip as well as off-chip DRAM memory connected via DDR interface.

5.5.1 On-Chip vs. Off-Chip Memory Accesses

There are two main ways to store data on FPGA board: (i) *off-chip* DRAM with several Gigabytes of capacity but high response delay and (ii) *on-chip* BRAM with few megabits

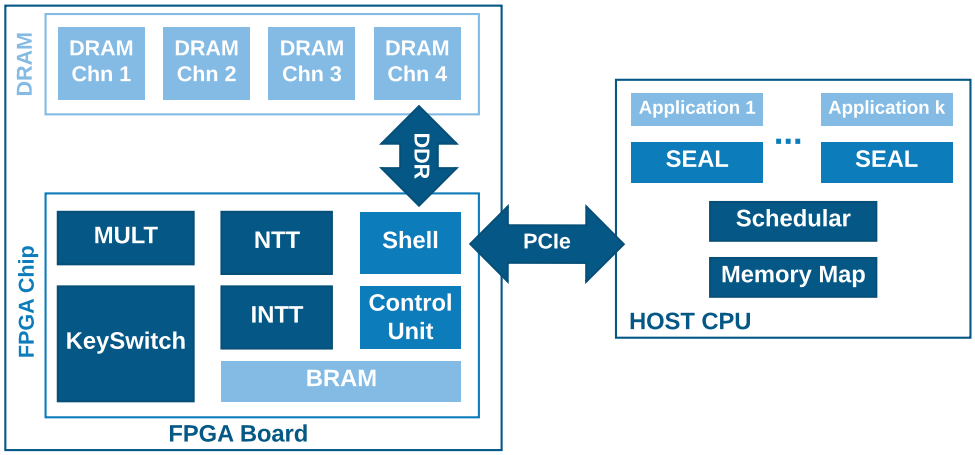


Figure 5.7: System-view of HEAX.

of capacity but very fast response time and high throughput. As has been shown by prior art [RJV+18, RTJ+19], leveraging off-chip memory to store intermediate results significantly reduces the overall performance due to high delays between subsequent reads and writes. One of our primary design goals is to avoid off-chip memory access as much as possible. We have introduced several techniques to use minimal on-chip memory, re-use many BRAM units, together with data compaction (see Section 5.3 and Section 5.4). As a result, no off-chip memory access is performed for $n = 2^{12}$ parameter set on both Arria 10 and Stratix 10 FPGAs; which is one of the main reasons for our unprecedented performance improvements.

For $n = 2^{13}$ parameter set, there is sufficient on-chip memory on Stratix 10 chip. Unfortunately, for $n = 2^{14}$, there is not enough BRAM available for our design, and as a result, we have to move some part of the data to off-chip memory. In order to minimize the effect of off-chip memory accesses, we choose to put key switching keys (`ksk`) in DRAM because of two main reasons: (i) the size of these keys grow very rapidly with HE parameters. In general, the size of `ksk` grows as $O(nk^2)$, and roughly speaking, k grows linearly with n which results in (almost) $O(n^3)$. This is the highest growth rate compared to all other memory components, including twiddle factors which grow as $O(nk)$. (ii) `ksk` is only read once per each `KeySwitch`. Note that each unique element of twiddle factors is read k times during one KeySwitch operation; thus, twiddle factors are less suitable candidates.

We distribute the `ksk` among four different DRAM banks such that at any point in time, the full capacity of off-chip memory bandwidth is used. In order to further mask the effect of DRAM accesses, we leverage the *burst mode* in which a long sequence of data is read at the same time on each channel. The entire process of reading `ksk` from DRAM is pipelined to minimize the drop in throughput of KeySwitch. It is worth-mentioning that DRAM bandwidth is sufficient to match the throughput of KeySwitch. Per each `KeySwitch`, two sets of `ksk` have to be streamed to FPGA chip. Each of these sets, hold $k \cdot (k+1)$-many vectors of size n. Substituting $n = 2^{14}$, $k = 8$, and 64-bit per each word results in approximately 151 megabits. We have to stream this volume of data in 383 microseconds (please see Table 5.8). Therefore, DRAM bandwidth should be higher than 49.28 GBps, which is indeed lower than the measured bandwidth of all four channels combined.

In addition to storing `ksk`, we use DRAM for one more purpose. In some applications, it is more efficient to store the result of computation in DRAM instead of sending them back to CPU (in case these results are going to be used soon). The address at which the result is stored is held on the CPU side and is shown as "Memory Map". The memory map is used to point to the ciphertext(s) that are stored in DRAM to be used later on without involving PCIe.

5.5.2 Data Transfer on PCIe

In order to maximize the utilization of computation blocks on FPGA, we need to interleave computation and data transfer between FPGA and CPU. We divide this design process into two parts: CPU-side and FPGA-side. On the CPU-side, we need to sequence and batch multiple operations in the program (that uses SEAL) and start the data transfer process on PCIe using *multiple threads*. On the FPGA-side, we need to allocate the necessary buffers to store the received data. In what follows, we explain these two parts in more detail.

Sequencing and Batching. Transferring data on PCIe involves three main steps: (i) a `memcpy` is issued to copy the content of the polynomial to pinned memory pages, (ii) CPU

signals FPGA that the data is ready, and (iii) FPGA reads the data from PCIe. In order to reduce the data copy time, Direct Memory Access (DMA) is used. However, even by relying on DMA, the maximum throughput that PCIe can provide depends on the message size and the number of simultaneous data transfer requests. Therefore, we transfer (at least) a complete polynomial ($2^{15} - 2^{17}$ Bytes) in each request. Moreover, we implement a multi-threaded data transfer mechanism that uses eight threads to interleave eight separate polynomials at a time to maximize the PCIe throughput and avoid unnecessary bubbles in the computation pipeline.

Double and Quadruple Buffering. For the MULT module, it suffices to double-buffer the input such that CPU writes to one of these buffers and FPGA reads from the other one. For KeySwitch module, however, we need to perform quadruple buffering due to the data dependency on input polynomial as discussed in Section 5.4. In order to make sure buffers are not overridden before they are read, we stop the writing process if the buffer has not been read yet.

5.6 Implementation and Experiments

5.6.1 Experimental Setup

In this section, we discuss the resource consumption of HEAX components as well as the performance comparison with CPUs and GPUs. To illustrate the adaptability of HEAX, we implement HEAX on two FPGAs which represent two different classes of computational resources. Table 5.1 summarizes the breakdown of resources of each FPGA chip. There are three major types of resources that are available:

• Digital Signal Processing (DSP) units that are able to perform one 27-bit or two 18-bit multiplications.

• Adaptive Logic Modules (ALM) are core logic units with two combinational adaptive look-up tables, a two-bit full adder, and four 1-bit Registers (REG).

• Block RAM (BRAM) units that are on-chip memories. Each M20K unit of BRAM holds

512-many 40-bit values.

Table 5.1: Summary of FPGA boards' specifications.

Board	Chip	Chip Resources					DRAM	
		DSP	REG	ALM	BRAM		#chnl.	BW (GBps)
					bits	#M20K		
Board-A	Arria 10 GX 1150	1518	1.71M	427K	53Mb	2.7K	2	34
Board-B	Stratix 10 GX 2800	5760	3.73M	933K	229Mb	11.7K	4	64

5.6.2 FHE Parameters and Security Guarantees

The security guarantees of HEAX directly derives from the CKKS scheme [CKKS17] since the functionality of the scheme is not altered. The security parameters for which we have instantiated HEAX are borrowed from the HE security standards [ACC+18] for 128-bit classical security. Changing the underlying word-size in HEAX reduces the number of DSPs used but does not affect the security since the total bitwidth of the ciphertext modulus is preserved [ACC+18]. Similarly, we leveraged the RNS-level parallelism which is proven to be secure [CHK+19].

We evaluate our design on a wide range of FHE parameters: from ciphertext polynomial size (n) of 2^{12} and 109-bit ciphertext modulus ($\lfloor \log qp \rfloor + 1$) to 2^{14} with 438-bit ciphertext modulus. We refer to these parameter sets as Set-A, Set-B, and Set-C, respectively (summarized in Table 5.2). Recall that k is the number of small RNS components of ciphertext modulus. Parameters with 128-bit post-quantum security require slightly smaller ciphertext moduli. We select as few prime moduli for RNS as possible for superior performance [HPS19]. Note that parameter sets corresponding to 2^{11} (or lower) are almost never used in practice due to the multiplication depth of 1 (or zero). Choosing 2^{15} (or higher) results in enormous computation blow-up and are also rarely used in practice.

Table 5.2: The HE parameter sets used in this work. n is the ciphertext polynomial size, qp is the ciphertext modulus, and k is the number of RNS components of q.

HE Param. Set	n	$\lfloor \log qp \rfloor + 1$	k
Set-A	2^{12}	109	2
Set-B	2^{13}	218	4
Set-C	2^{14}	438	8

5.6.3 Resource Consumption

Computation Cores. Table 5.3 provides a detailed resource consumption of Dyadic, NTT, and INTT computation cores as well as the number of pipeline stages (delay) for each core.

Table 5.3: Resource consumption of each computation core.

Core Name	DSP	REG	ALM	#Stages
Dyadic	22	4526	1663	23
NTT	10	6297	2066	50
INTT	10	5449	2119	49

Basic Modules. Table 5.4 provides a detailed resource consumption of different modules (with various number of cores). The BRAM utilization is reported for Set-B parameters ($n = 2^{13}$). The BRAM *bits* usage in each module does not depend on the number of cores but the number of M20K units does. The reason is that more coefficients are stored in parallel M20K units. In the last column, the number of cycles that takes for each module to process a polynomial (or pair of polynomials in case of MULT module) is reported.

Table 5.5: KeySwitch architecture for different HE parameter sets.

FPGA Device	HE Param. Set	KeySwitch Architecture Parameter Set
Arria10	$n = 2^{12}$ (Set-A)	$1 \times \text{INTT}^{(8)} \rightarrow 2 \times \text{NTT}^{(8)} \rightarrow 3 \times \text{Dyad}^{(4)} \rightarrow 2 \times \text{INTT}^{(4)} \rightarrow 2 \times \text{NTT}^{(8)} \rightarrow 2 \times \text{MS}^{(2)}$
Stratix10	$n = 2^{12}$ (Set-A)	$1 \times \text{INTT}^{(16)} \rightarrow 2 \times \text{NTT}^{(16)} \rightarrow 3 \times \text{Dyad}^{(8)} \rightarrow 2 \times \text{INTT}^{(8)} \rightarrow 2 \times \text{NTT}^{(16)} \rightarrow 2 \times \text{MS}^{(4)}$
	$n = 2^{13}$ (Set-B)	$1 \times \text{INTT}^{(16)} \rightarrow 4 \times \text{NTT}^{(16)} \rightarrow 5 \times \text{Dyad}^{(8)} \rightarrow 2 \times \text{INTT}^{(4)} \rightarrow 2 \times \text{NTT}^{(16)} \rightarrow 2 \times \text{MS}^{(4)}$
	$n = 2^{14}$ (Set-C)	$1 \times \text{INTT}^{(8)} \rightarrow 4 \times \text{NTT}^{(16)} \rightarrow 5 \times \text{Dyad}^{(8)} \rightarrow 2 \times \text{INTT}^{(1)} \rightarrow 2 \times \text{NTT}^{(8)} \rightarrow 2 \times \text{MS}^{(4)}$

Table 5.6: Resource consumption of HEAX for different HE parameter sets.

FPGA Device	HE Param. Set	DSP (%)	REG (%)	ALM (%)	BRAM bits (%)	BRAM #M20K (%)	Freq. (MHz)
Arria10	Set-A	1185 (78)	723188 (42)	246323 (58)	26596320 (48)	1731 (64)	275
Stratix10	Set-A	2018 (35)	1554005 (42)	582148 (62)	26907592 (11)	3986 (34)	300
	Set-B	2610 (45)	1976162 (53)	698884 (75)	201332624 (84)	10340 (88)	300
	Set-C	2370 (41)	1746384 (47)	599715 (64)	182847524 (76)	9329 (80)	300

Table 5.7: Performance comparison of HEAX with CPU. Number of operations per second for CKKS *low-level* operations.

FPGA Device	HE Param. Set	NTT			INTT			Dyadic MULT		
		CPU	HEAX	Speed-up	CPU	HEAX	Speed-up	CPU	HEAX	Speed-up
Arria10	Set-A	7222	89518	12.4	7568	89518	11.8	36931	1074219	29.1
Stratix10	Set-A	7222	195313	27.0	7568	195313	25.8	36931	1171875	31.7
	Set-B	3437	90144	26.2	3539	90144	25.5	18362	585938	31.9
	Set-C	1631	41853	25.7	1659	41853	25.2	9117	292969	32.1

Table 5.4: Resource consumption of basic modules.

Module	#Cores	DSP	REG	ALM	BRAM #bits	BRAM #M20K	Cycles
A10 Shell	-	1	79203	39222	886496	144	-
S10 Shell	-	2	86984	45612	1201096	173	-
MULT	4	88	42817	15795	1104384	65	1024
	8	176	61878	22160		65	512
	16	352	93594	35257		164	128
	32	704	181503	62157		293	64
NTT	4	40	61670	22316	1514496	86	6144
	8	80	96919	36336		185	3072
	16	160	196205	67865		380	1536
	32	320	387357	142300		725	768
INTT	4	40	63917	22700	1514496	86	6144
	8	80	104575	37331		185	3072
	16	160	182478	68645		380	1536
	32	320	384267	144957		724	768

Complete Design. Table 5.6 provides a breakdown of FPGA resource consumption for

Table 5.8: Performance comparison of HEAX with CPU. Number of operations per second for CKKS *high-level* operations.

FPGA Device	HE Param. Set	KeySwitch			MULT+ReLin		
		CPU	HEAX	Speed-up	CPU	HEAX	Speed-up
Arria10	Set-A	488	44759	91.7	420	44759	106.6
Stratix10	Set-A	488	97656	**200.5**	420	97656	**232.5**
	Set-B	97	22536	**232.3**	84	22536	**268.3**
	Set-C	16	2616	**163.5**	15	2616	**174.4**

different HE parameter sets. The complete design encompasses the KeySwitch module along with the MULT module. For standalone NTT requests from CPU, the NTT modules within KeySwitch is used.

5.6.4 Performance

Critical Paths and Maximum Clock Frequency. We have analyzed the critical paths of our design and have eliminated such paths during many design iterations reaching the maximum clock frequency of 275 MHz and 300 MHz for Arria 10 and Stratix 10 FPGA chips, respectively.

Scalability. One of design principles of HEAX is that it can automatically be instantiated at different scales with no manual tuning, enabling cloud providers to seamlessly use HEAX based on the underlying hardware resource. To illustrate this, we have instantiated HEAX for the *same HE parameters* (Set-A) but at two *different scales* (see Table 5.5). The up-scaled version on Stratix 10 consumes (close to) twice the resources (Table 5.6) and provides twice the throughput compared to Arria 10 instantiation (see Table 5.8).

Performance Comparison with GPUs. To the best of our knowledge, there does not exist any work based on FPGAs or GPUs for CKKS scheme. In Table 5.9, we compare the performance of our "NTT architecture" on Stratix10 (which holds ten 16-core NTT modules) with two NVIDIA GPUs [ABVMA18]. Not only HEAX consumes significanlty less power but it is **36–81**$\times$ faster compared to data-center GPUs.

Table 5.9: Performance comparison (operations per second) of HEAX with NVIDIA GPUs for NTT computation.

Polynomial Size	HEAX (10×16-cores)	Tesla-K80 (2496-cores)	Tesla-P100 (3584-cores)	Performance Comparison
2^{12}	1953130	25641	27777	**70–76**×
2^{13}	901440	20833	25000	**36–43**×
2^{14}	418530	5181	11494	**36–81**×

Performance Comparison with CPUs. We compare the performance of HEAX with Microsoft SEAL V3.3 [SEA19], which is an FHE library for BFV and CKKS schemes that has undergone several years of performance optimizations. We measure the performance of SEAL on a single-threaded Intel Xeon(R) Silver 4108 running at 1.80 GHz; which is a similar CPU used in prior art [RTJ+19]. The single-thread baseline is used by prior art for measuring the performance (non-CKKS schemes) [RTJ+19]. In addition, SEAL is thread-safe but *not multithreaded* due to the complex data dependencies, hence, we cannot compare to a multi-threaded execution. In general, CKKS evaluation functions do not have a balanced parallelizable computation flow and many parts are not parallelizable at all. For instance, the "Modulus Switching" is not parallelizable leading to the Data-Dependency 2 (Figure 5.6). This is the reason why we cannot allocate a single `NTT`/`INTT` module in `KeySwitch` and use it over time for different steps. Instead, we design an end-to-end *pipelined* design and use the chip-area proportional to the computation overhead.

Table 5.7 shows the performance results (number of operations per second) of HEAX for low-level operations and its comparison with SEAL. Results are reported for processing a single polynomial (in case of `NTT`/`INTT`) or pair of polynomials (`MULT`). On Stratix 10, 16-core modules are instantiated. On Arria 10, a 16-core `MULT` and 8-core `NTT`/`INTT` modules are used (see Table 5.5). Note that we report the performance results for low-level operation merely for completeness. These operations are rarely used in isolation and are instead used as part of high-level operations. For high-level operations, i.e., Rotation and Relinearization (using `KeySwitch`) and a complete ciphertext multiplication (using `MULT` and `KeySwitch`), the

performance improvements are more pronounced as shown in Table 5.8. As can be seen, HEAX achieves close to *two orders of magnitude* performance improvement using Arria 10 compared to CPU (first row of Table 5.8). On a more powerful FPGA, i.e., Intel Stratix 10 (Board-B), HEAX achieves **164–268**$\times$ performance improvements among various HE parameter sets (second to fourth rows of Table 5.8).

5.7 Related Work

The CKKS scheme is one of the most recently proposed FHE schemes that allows homomorphic operations on fixed-point numbers; making it the prime candidate for machine learning applications. To the best of our knowledge, no hardware architecture has been proposed for the CKKS scheme, and in this paper, we propose the first of its kind. As a result, it is not fair to compare the performance of HEAX with previous designs that focus on non-CKKS schemes. In what follows, we briefly review the research effort related to FPGA, ASIC, and GPU-based acceleration for non-CKKS schemes.

Hardware Acceleration for non-CKKS Schemes. In [RJV+18], a system based on FPGA is proposed for BFV scheme to process ciphertext polynomial sizes of 2^{15}. However, due to the massive off-chip data transfer, their design does not yield superior performance compared to CPU execution.

Perhaps, the closest work to ours is by Roy et al. [RTJ+19] in which authors propose an architecture for BFV scheme and implement their design on Xilinx Zynq UltraScale+ MPSoC ZCU102. In order to avoid off-chip memory accesses, authors focus on $n = 2^{12}$ ciphertext sizes and report 13$\times$ speed-up (using two instances of their proposed processors) compared to the FV-NFLlib [FV-] executing on an Intel i5 processor running at 1.8 GHz. However, compared to a more optimized Microsoft SEAL library [SEA], FV-NFLlib is 1.2$\times$ slower [BVMA18]. In addition, our design is significantly more modular and scalable. We have instantiated HEAX for

three different set of HE parameters with no manual tuning (polynomial sizes of 2^{12}, 2^{13}, and 2^{14}). Moreover, HEAX has a multi-layer pipelined design and is drastically more efficient, offering more than two orders of magnitude performance improvement compared to Microsoft SEAL running on Intel Xeon Silver 4108 at 1.8 GHz (note that similar processor is used compared with [RTJ+19] running at identical frequency).

FPGA-based Co-Processors. Designing co-processors has also been studied in the literature. These co-processors work in conjunction with CPUs and accelerate one or more of the homomorphic operations [JGCM+15, MSR+17, CRS16, HB18, MOHO14, KG19]. In [MSR+17] and [HB18, MOHO14], authors focus on designing hardware architecture for the *encryption* operation only, by leveraging Karatsuba and Comba multiplication algorithms, respectively. In [CRS16], a Homomorphic Encryption Processing Unit (HEPU) is proposed for LTV scheme [LTV12]. Authors focus on accelerating the Chinese Remainder Transform (CRT) for power-of-2 cyclotomic rings and report 3.2–4.4× performance improvements for homomorphic multiplication using Xilinx Virtex-7.

Large-Integer Multiplication Hardware Acceleration. A line of research focuses on designing very large integer multipliers (768K-bit to 1.18M-bit multiplications) – based on FPGAs or ASICs – that can be used to accelerate homomorphic operations [WH13, WHEW13, DÖS13, DÖS14a, CMO+13]. In [CMO+14], a large-integer multiplier and a Barrett modular reduction are proposed that can accelerate HE operations by 11×.

GPU-based Acceleration. GPU is an alternative computing platform to accelerate evaluation functions [DS16, cuF, nuFer, BPA+19, KGV16, WHC+12]. Wang et al. [WHC+12] have proposed the first GPU acceleration of FHE that targets Gentry-Halevi [GH11] scheme. Subsequent improvements are reported in [WHC+13]. In [WCH14], a GPU-based implementation of BGV scheme [BGV12] is introduced. In [BPA+19], a comprehensive study is reported for multithreaded CPU execution as well as GPU for the BFV scheme. To the best of our knowledge, there is no GPU-accelerated implementation of the CKKS scheme. GPUs normally offer less

performance per watt of power than FPGAs by design. Therefore, FPGAs are more suitable candidates for high-performance and low-power secure computation.

Acceleration of YASHE and LTV Schemes. Several works [ÖDSS16, DDS14, DÖSS15, ÖDSS15, CGRS14, CRS16] focus on improving the performance of YASHE [BLLN13] and LTV [LTV12] schemes or their variants. These constructions – based on an overstretched NTRU assumption – are subject to a subfield lattice attack [ABD16] and are no longer secure. In [RJV+15], an architecture for YASHE scheme is proposed that provides 25× performance improvement over CPU. However, authors assume unlimited memory bandwidth which renders off-chip memory accesses free of cost and is not a realistic assumption. Pöppelmann et al. [PNPM15] have also proposed an architecture for YASHE scheme. Since ciphertexts are prohibitively large to be stored on on-chip memory, authors propose to leverage the idea of Cached-NTT [Baa99, Baa05] to reduce off-chip memory accesses. In contrast, HEAX relies on the ring isomorphism property and perform independent computation on RNS components. This, in turn, allows us to avoid off-chip memory accesses for small HE parameters and minimize such accesses for large parameters.

Hardware Acceleration of Secure Computation Protocols. Secure Multi-Party Computation (SMPC) protocols can also be used for the task of privacy-preserving MLaaS. The Garbled Circuits protocol [Yao86a] is one of the secure *two-party* computation protocols for which FPGA-based accelerators have been proposed [HK19, FIL17, HRGK18]. However, compared to HE, the communication between parties is significantly higher, making the end-to-end protocol mostly bandwidth constrained. Thus, accelerating the *computation* portion of SMPC protocols does not necessarily reduce the execution time of the protocol in general. In contrast, the main bottleneck of HE is the computation overhead and reducing the homomorphic evaluation time directly increases the practicality of HE-based computations.

5.8 Summary

In this chapter, we introduced a novel set of architectures for Fully Homomorphic Encryption (FHE). To the best of our knowledge, HEAX is the *first* architecture and fully-implemented hardware acceleration for the CKKS FHE scheme. CKKS is the prime candidate for machine learning on encrypted data due to floating-point support of this scheme. The components designed in HEAX can also be used for other lattice-based cryptosystems and other FHE/HE schemes. The proposed architecture provides a unique degree of flexibility that can be readily adjusted for various FPGA chips. As a proof-of-concept, we have implemented HEAX on two different FPGAs with contrasting hardware resources. Moreover, unlike prior FPGA-based acceleration for BFV scheme, our design is not tied to a specific FHE parameter set. We evaluate HEAX on a wide range of FHE parameters demonstrating more than *two orders of magnitude* performance improvements. We hope that HEAX paves the way for large-scale deployment of privacy-preserving computation in clouds.

5.9 Acknowledgements

This chapter, in part, has been published at the Proceedings of 2020 ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) and appears as: M. Sadegh Riazi, Kim Laine, Blake Pelton, Wei Dai, "HEAX: An Architecture for Computing on Encrypted Data" [RLPD20]. The dissertation author was the primary author of this material.

Chapter 6

MPCircuits: Compact Boolean Circuits for Secure Multiparty Computation

Secure Multi-party Computation (MPC) is one of the most influential achievements of modern cryptography: it allows evaluation of an arbitrary function on private inputs from multiple parties without revealing the inputs. A crucial step of utilizing contemporary MPC protocols is to describe the function as a Boolean circuit. While efficient solutions have been proposed for special case of *two-party* secure computation, the general case of more than two-party is not addressed. This chapter proposes `MPCircuits`, the *first automated* solution to devise the optimized Boolean circuit representation for any MPC function using hardware synthesis tools with new customized libraries that are scalable to multiple parties. `MPCircuits` creates a new end-to-end tool-chain to facilitate practical scalable MPC realization. To illustrate the practicality of `MPCircuits`, we design and implement a set of five circuits that represent real-world MPC problems. Our benchmarks inherently have different computational and communication complexities and are good candidates to evaluate MPC protocols. We also formalize the metrics by which a given protocol can be analyzed. We provide extensive experimental evaluations for these benchmarks; two of which are the *first* reported solutions in multi-party settings. As our experimental results

indicate, `MPCircuits` reduces the computation time of MPC protocols by up to 4.2×.

6.1 Introduction

Secure multi-party computation (MPC a.k.a., SMC) provides a provably-secure method for multiple parties to jointly evaluate a function on their private inputs without disclosing the input values to each other. MPC protocols can be categorized into two main groups: protocols based on (i) the GMW (Goldreich-Micali-Wigderson) paradigm [GMW87] and (ii) the Garbled-Circuit (GC) paradigm [Yao86b]. The original idea of two-party GC is later generalized for multi-party setting in the Beaver-Micali-Rogaway (BMR) protocol [BMR90]. Both paradigms require the underlying function to be represented as a *Boolean circuit*. The tools and methods for Boolean computations of two-party protocols are available, but they are not readily scalable or available for multiple parties. Present ad-hoc realization of secure multi-party tasks do not provide a holistic tool usable for a variety of other MPC applications.

Two standing challenges that users face while utilizing MPC protocols are: (i) generating optimized Boolean circuits for the pertinent task, and (ii) the necessity of knowing the details of the protocol. The first complication often results in a high inefficiency in the protocol execution. The latter is even more critical, triggering possible security breaches if the exact protocol is not followed. Protocols that interpret a multi-party computation as multiple invocations of two-party secure computation are not just impractical, but also specifically susceptible to such breaches. Therefore, there is a need for an end-to-end solution that bridges the gap between usability and secure realization of MPC protocols.

During the past two decades, a number of practical realizations for the special case of *two-party* secure computation have been presented, reducing the execution time by several orders of magnitude [MGC+16, LWN+15, ZE15, BHKR13, SHS+15, RWT+18b, RSK17b, RDGK16a]. However, less focus has been on the general problem of secure *multi-party* (more than two)

computation, even though many practical problems (e.g., auction and voting) are inherently multi-party and cannot be reduced to two-party settings.

To the best of our knowledge, there have only been three MPC realizations that support more than two parties [BELO16, CHK+12, BDNP08]. Among them, only [BDNP08] supports circuit generation: the crucial first step of MPC. However, this framework precedes one of the most crucial MPC optimizations: free-XOR [KS08b], which allows `XOR`, `XNOR`, and `NOT` gates to be evaluated without any communication or encryption. The two more recent frameworks [BELO16, CHK+12] support free-XOR but only focus on the specific (ad-hoc) protocol execution with no systematic solution for the circuit generation step.

In this chapter, we present the *first automated* methodology to generate Boolean circuits, customized for MPC protocols with state-of-the-art optimizations. Inspired by TinyGarble [SHS+15], the most efficient Boolean circuit generator for the two-party setting, we leverage standard logic synthesis tools for this purpose. Note that two-party libraries such as TinyGarble cannot be used for the MPC problem since the synthesis technology libraries are not compatible with the MPC protocols. In addition, the order of logic computation (determined by the API and the Boolean netlist sorter) is radically different for two-party protocols. `MPCircuits` relies on designing new technology libraries for the logic synthesis tools customized for MPC protocols. Our solution can be integrated with any cryptographic back-end engine for the MPC protocol, e.g., the realizations in [BELO16, CHK+12], to allow users to perform a holistic secure multi-party computation.

This work also aims to facilitate future research on MPC. One of the key enablers in analyzing different protocols is to have a comprehensive set of benchmarks representing different applications/tasks. To do so, we compile a set of benchmarks that represent critical real-world MPC problems. Our benchmark set includes practical problems of auction, voting, and set intersection that have been evaluated in prior MPC literature. However, the methodologies presented in this chapter for auction and voting do not need the presence of any trusted third parties as in the

existing work [BDJ+06,CCM08]. We also design circuits for two new benchmarks, namely, stable matching and nearest-neighbor search. Our circuits for stable matching and nearest-neighbor search are the *first* solutions in the multi-party setting. Each proposed benchmark captures a different set of requirements and domains, which ensures the applicability of `MPCircuits` to diverse scenarios.

We use a set of metrics to quantify the performance of `MPCircuits` on each of the benchmarks. These metrics encompass different characteristics of the MPC protocol, analyzing the performance of the solution for settings with varying computation power and communication bandwidth. We prototype our solution and perform extensive evaluations on our benchmarks for a range of parameter sizes. In brief, our three main contributions are as follows:

- Introducing the first holistic solution to automatically create optimized Boolean circuits for MPC. We provide a new technology library for hardware synthesis tools to generate circuits compatible with MPC. Our approach is modular and can produce efficient circuits for various functionalities.

- Designing and implementing circuits for five compelling secure multi-party computation tasks. These tasks represent five key MPC problems that cover most of the applications suggested in literature, namely, auction, voting, set intersection, stable matching, and nearest neighbor search. We further introduce metrics by which each benchmark could be analyzed for efficiency and scalability.

- Creating automated tools for end-to-end MPC realization, i.e., a simple-to-use API and interpreter programs to convert the generated netlist to formats usable by MPC back-end engines. Extensive experimental results are provided for the different parameter configurations confirming the efficiency and scalability of `MPCircuits`.

Code Availability: `https://github.com/sadeghriazi/MPCircuits`

6.2 Automated Circuit Generation

The first, and one of the most critical, step in practical secure realization of a function through MPC protocols is generation of the Boolean circuit that describes the pertinent functionality. We now elaborate on our methodology for automatically creating the Boolean circuit such that it is optimized for MPC as well as making it compatible with any given realization of the protocol. The corresponding steps of the circuit generation in `MPCircuits` are illustrated in Figure 6.1.

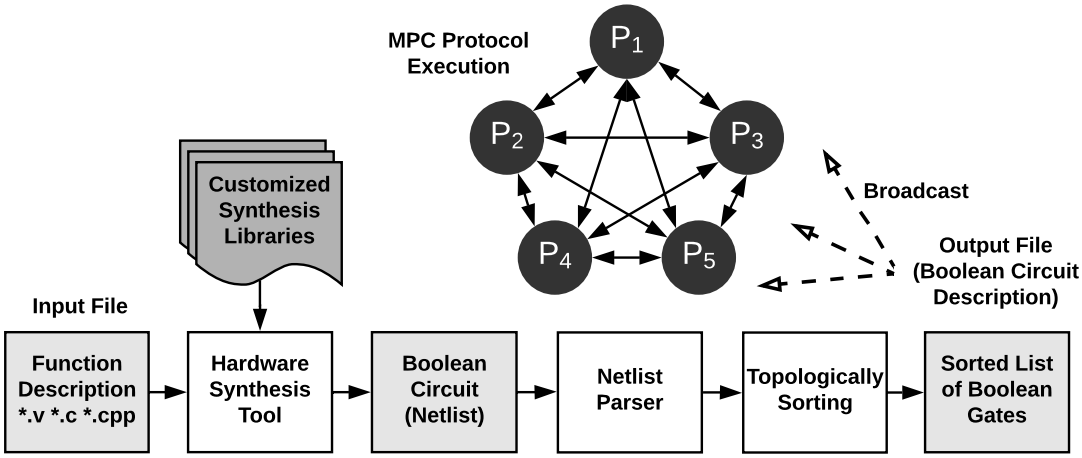


Figure 6.1: Global flow of `MPCircuits` circuit generation.

In the first step, the user writes the function description in a Hardware Description Language (HDL), e.g., Verilog. With recent progress in High-Level Synthesis (HLS) tools, it is also possible to develop the function in C and convert it to HDL using these tools. Inspired by TinyGarble [SHS+15], we employ logic synthesis tools to compile the HDL source code using our customized libraries. Thus, we generalize the idea of TinyGarble for multi-party (more than two-party) secure execution. Recall that in contrast to non-`XOR` gates, `XOR` gates do not require communication or encryption during garbling/evaluation due to the free-`XOR` technique [KS08b]. As illustrated in [BELO16], the number of non-XOR gates determines the computation and communication cost. Therefore, `MPCircuits` objective function is to minimize the *total number of non-XOR gates* ($n_{non-\texttt{XOR}}$) in the circuit description.

We re-define the problem of minimizing $n_{non-\texttt{XOR}}$ as a special case of logic optimization to be performed by the synthesis tools. These tools have been subject to more than two decades of improvement leading to sophisticated algorithms to minimize a given constraint, e.g., power consumption, circuit area, or critical path. We develop new synthesis and cell libraries to be utilized by such logic synthesis tools. Our synthesis libraries incorporate the realization of basic arithmetic and logic operations (add, subtract, multiplication, division, if-else, etc.) using minimum number of non-`XOR` gates. The cell library in `MPCircuits`, which is used for ASIC mapping, contains Boolean logic gates `AND`, `shifted_AND`, `XOR`, and `XNOR`. The area of the `XOR` and `XNOR` gates are set to zero and that of the other gates to one and the constraint is set to *minimizing the total area of the circuit*. As a result, the synthesis process minimizes the total number of non-`XOR` gates in the final netlist. This process *automatically* generates an optimized Boolean circuit for the BMR protocol. Note that the output of the TinyGarble framework cannot be used for the MPC protocols since the produced netlist contains logic gates that are not supported by MPC realizations [BELO16].

The logic synthesis tool outputs a standard Verilog netlist containing cells that are included in the cell library. In order to use the netlist in a MPC protocol, one has to perform certain post-synthesis steps. This translation involves three steps: (i) a parser reads the Verilog netlist and converts it to an array of structures defining the logic operation of a particular gate and its input/output connections, (ii) a scheduler reads the connection information of the gates and topologically sorts them based on the dependencies, (iii) the sorted array of gates is written to a file in a specific format. This file is public and is sent to all the parties participating in the secure protocol. Note that, only the last step is specific to one particular MPC realization (BMR protocol in our implementation). Therefore, our methodologies can be applied to a large variety of secure protocols that are based on a Boolean netlist (e.g., the framework in [CHK$^+$12]) with a simple modification to the last step.

Our proof-of-concept implementation is based on the BMR realization presented

in [BELO16]. To the best of our knowledge, this is one of the only two MPC frameworks that supports the free-XOR [KS08b] optimization. Moreover, authors have optimized the BMR protocol to have a *constant number of rounds* which is strongly preferred in Internet settings where the communication delay is significant. In [BELO16], most of the computation can be shifted to the offline phase and precomputed in advance. We utilize the SCAPI [SCA17] library (source code of [BELO16]) in our implementation. This library supports only five types of Boolean logic. Therefore, our cell library contains only those five gates. However, this constraint does not increase the number of non-XOR gates and only increases the number of XOR gates. Incorporating the other gates in the cell library is straightforward, in case they are supported by another realization of an MPC protocol.

Security Model. The BMR protocol in [BELO16] is provably secure in the Honest-but-Curious (HbC) security model. In this model, all participating parties follow the protocol but they can attempt to extract more about the other parties' input from the information they send and receive. There are two variants of HbC adversary model: honest-majority and dishonest-majority. In the former model, the protocol is secure as long as the majority of the parties are not corrupted. In the latter model, *any* number of corrupted parties cannot infer information other than what can be inferred from the outcome of the computation. For example, the FairplayMP [BDNP08] framework is secure for an honest majority. In addition, protocols in HbC model are generally orders of magnitude faster than the ones in *malicious* model [BELO16] in which parties can deviate from the protocol anytime. HbC model is the building block for stronger security models such as security against malicious adversaries. `MPCircuits` can automatically generate optimized Boolean circuits for both security models. Note that the methodologies presented in this work do not change the security or correctness of the MPC protocol. `MPCircuits` provides an automated solution to generate Boolean circuit representations that have minimal number of non-XOR gates, thus, improving the *efficiency* of the MPC protocol.

6.3 Methodology

We design and implement a set of Boolean circuits for five compelling MPC tasks and report the experimental results for all of them using our circuit synthesis approach. Three of these applications, i.e., Auction, Voting, and PSI have been studied in the prior literature and we compare `MPCircuits` with state-of-the-art solutions. In `MPCircuits`, we have provided the *first secure solution* for K-nearest neighbor search (K-NNS) and Stable Matching. Not only do these two benchmarks address real-world needs, but each of them captures an important criteria: (i) in K-NNS (unlike other benchmarks) only one party receives the result of the protocol and therefore makes the process of Oblivious Transfer (OT) asymmetric since only one party evaluates the garbled circuit and only $(n-1)$ OTs are performed (compare to $(n-1)^2$); this is the first reported instance of an asymmetric OT. (ii) In Stable Matching, the size of the circuit grows with $O(n^4)$. In all other benchmarks the point-to-point communication rapidly becomes the bottleneck. Therefore, they cannot be used to validate other properties such as the scalability of the circuit generation. Evaluating an MPC framework on the Stable Matching benchmark illustrates the scalability of the framework in terms of circuit generation since the number of gates rapidly increases with the increase in the number of parties and soon becomes the bottleneck.

6.4 Auction

We design a Boolean circuit for auction in which the highest bidder is selected and pays the bid value. In traditional mechanisms, the auction is processed by a third party and all of the bid values are revealed to the third party. This raises many privacy concerns such as the possibility of information leakage or collusion between one of the bidders and the executing party. Here, we implement and analyze both types of auctions and illustrate the practicality and scalability of our solution.

Input: Each party P_i holds a bid_i, $i = 1...n$.

Output: The index (ID) of the highest bidder i_{max} and the highest bid value $x_{pay} = max(x_1, ..., x_n)$.

Parameter(s): Number of bits b to represent a bid.

6.4.1 Circuit Design

Figure 6.2 shows the internal architecture of the Boolean circuit for auction. The inputs to the circuit are the bids (upper side) and the output is the maximum bid value and the index (ID) of the winner. The darker blocks correspond to inputs and outputs of the circuit. Bids are compared in pairs using the comparison (`CMP`) blocks. The maximum value is then passed on to the next *layer* and so forth. A simple solution to compute the winner ID (WID) is to pass along the index of the higher bid at each stage. However, this solution requires $O\big(n\ lg(n)\big)$ `AND` gates where $lg(.)$ is the base-2 logarithm.

We propose a more optimized approach that requires $O\big(n\big)$ number of `AND` gates. While the complexity does not change significantly, in practice, the number of `AND` gates is reduced by a factor of $lg(n)$. Our method is based on re-using the output of the `CMP` blocks. Consider the last `CMP` block at the bottom right corner. Depending on the output of this block, one can identify the most significant bit (MSB) of the WID. For example, if the output is 1, it shows that the winner is from the second half of the bidders and hence, the WID starts with one. We can *recursively* continue this approach and depending on the already computed WID digits, the next digit is selected. More precisely, if we denote the output of the j^{th} `CMP` block at layer l by CMP[l][j], we have

$$\text{WID}[\alpha] = \text{CMP}[\alpha - 1][\text{WID}[\alpha - 1 : 0]]$$

At the end, WID holds the ID of the winner in reversed order (from least significant bit to

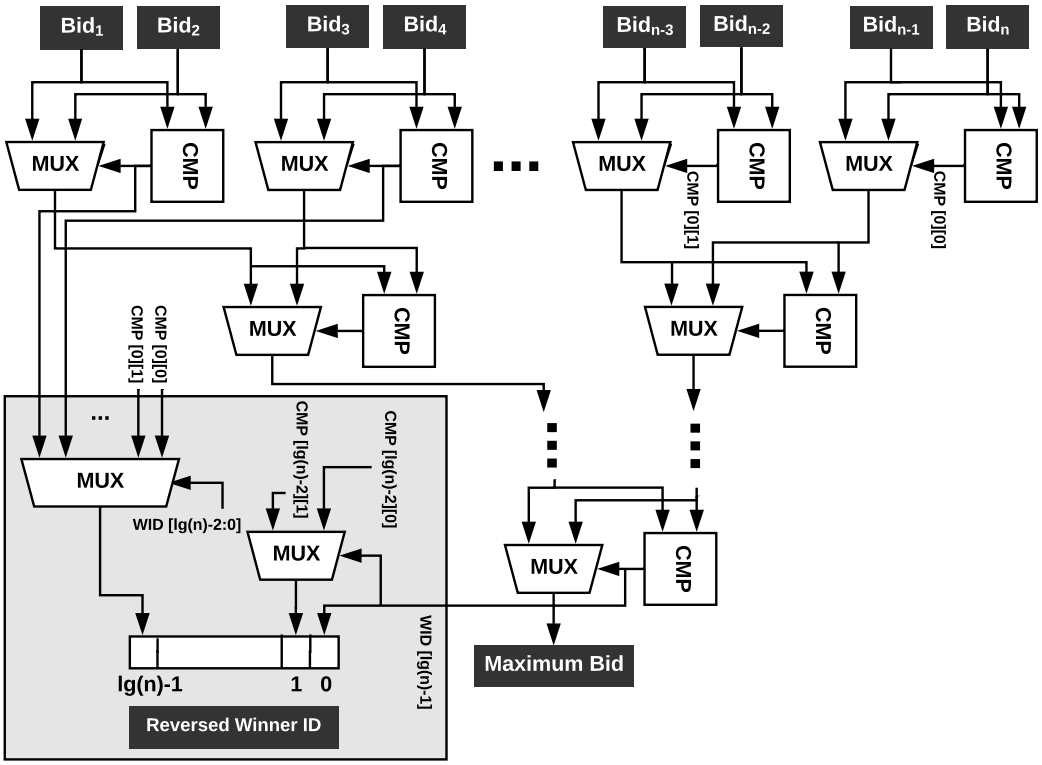


Figure 6.2: Boolean circuit for auction.

most). The complexity of the total number of AND gates in the circuit is $O(n\,b)$.

6.5 Voting

A secure voting mechanism can preserve the privacy of all voters in an election. This, in turn, can replace old solutions based on anonymization and law-enforcement. The aforementioned solutions are all centralized and have a single point of failure. A modern secure voting mechanism can ensure the correctness of the election and privacy of voters even in the presence of any set of corrupted parties.

Input: Each party P_i holds the index of the candidate to whom she wants to vote ($vote_i$).

Output: The index of the candidate, n_h, with the highest vote.

Parameter(s): Number of candidates n_c.

6.5.1 Circuit Design

The internal design of the Boolean circuit for voting is illustrated in Figure 6.3. The inputs to the circuit are n votes, each representing the index of a candidate ($lg(n_c)$-bit). Each vote is an input to a "$lg(n_c)$ to n_c" Decoder (DEC) module. Therefore, based on the vote value, only one of the output lines of the DEC is set to one. These output wires are then connected to a COUNT module to count the number of votes. The COUNT module is implemented as a binary tree of ADD blocks. At each level of the binary tree, the operands' bit-length of the ADD blocks are set to the minimal value that can accumulate the result. Hence, the COUNT module results in an efficient realization that only requires $x-1$ non-XOR gates for counting the number of ones in a binary array of size x. The final step is to find the maximum number of votes. This task can be implemented using the *auction* module (Section 6.4), where the bids are vote-counts. Finally, the output of the circuit is the ID of the winner candidate (WID).

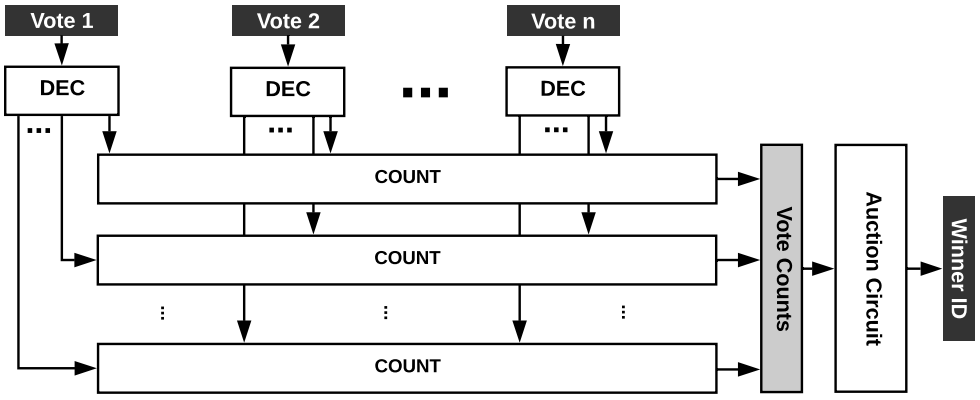


Figure 6.3: Boolean circuit for voting.

In case of a tie in the vote-counts, our circuit outputs the first candidate as the winner. If another tie-breaking mechanism is needed, our solution can easily be modified due to its modular structure. The complexity of the total number of AND gates in each of the circuit components are $O\big(n\ lg(n_c)\big)$ (decoders), $O\big(n\big)$ (COUNT modules), and $O\big(n_c\ lg(n)\big)$ (auction circuit). The overall circuit complexity is $O\big(n\ lg(n_c)+n_c\ lg(n)\big)$ which scales linearithmicly with n and n_c.

6.6 Set Intersection

Private Set Intersection (PSI) allows two or multiple parties to obtain the elements at the intersection of their sets without revealing the other elements that are not in common. For example, multiple people can identify their mutual contact profiles/friends by inputting their contact list to the PSI protocol without revealing the rest of their contact lists. At the end of the protocol, only the mutual list of all parties is revealed.

In E-commerce, an online advertisement agency and a company can participate in the PSI protocol where the advertisement agency inputs its list of all the people who have been shown the ads of the company. The second set of inputs to the protocol is the list of the people who have bought the products provided by the company. At the end of the PSI protocol, both entities know how many people have bought the product as a result of seeing the advertisement. This provides a way to understand the effectiveness of the advertisement for the company. Note that the same process could not be realized in plaintext due to various privacy/security reasons. Revealing such information is privacy invasive and can damage the reputation of both the companies. In addition, disclosing customer's data might be against the law in some situations.

Input: Party P_i holds a set $S_i \subset \Omega$ where Ω is the universal set.

Output: The intersection set $S = \cap_{i=1}^{n} S_i$.

Parameter(s): The size of the universal set Ω or equivalently the number of bits required to describe an element in the universal set $b = \lg|\Omega|$. Maximum number of elements in each party's set m.

6.6.1 Circuit Design

Two different implementations are provided for PSI: a Bitwise-AND based circuit and a Sort-Merge-Compare-Shuffle (SMCS) based circuit. The first one is more efficient for scenarios in which Ω is small whereas the second approach is more suitable when m is small and Ω can be

very large. Note that sets are represented differently in the two implementations as we explain in each section.

Bitwise-AND. In this implementation, each set is equivalent to a binary vector. The binary value at index j denotes the presence of the j-th element in a given set. Therefore, each set is represented as a $|\Omega|$-bit binary vector. The intersection set S is computed as *bit-wise* `AND` between all of the sets provided by all parties. As a result, the complexity of the circuit is $O(n\,|\Omega|)$, linear in both the number of parties and the size of the universal set; but *independent* from the number of elements in each parties' set m.

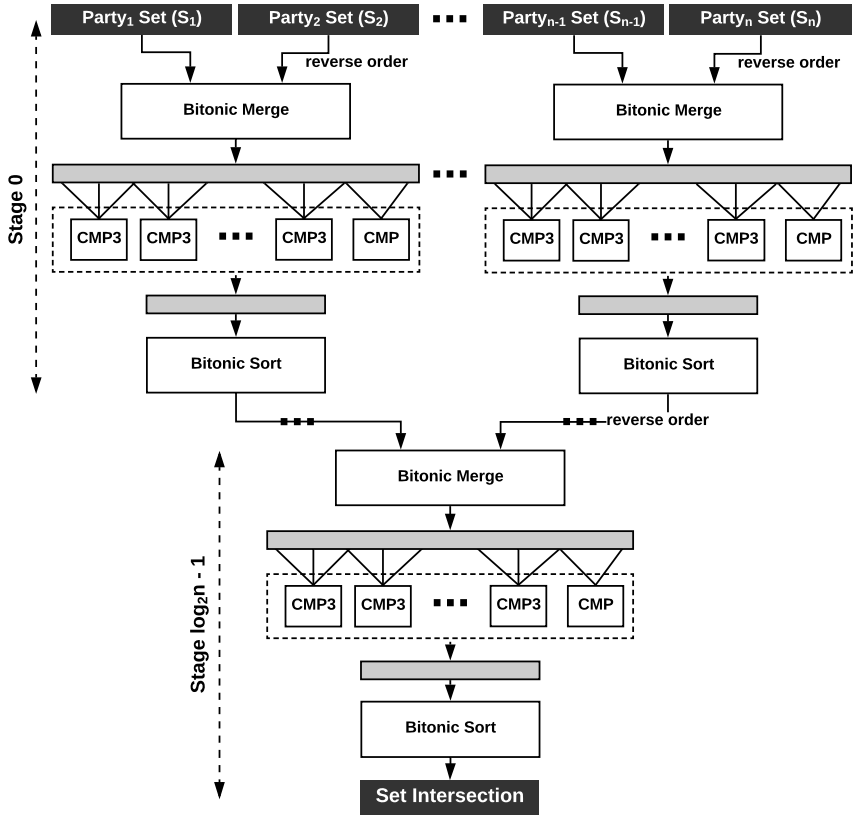


Figure 6.4: High-level circuit description of the Sort-Merge-Compare-Shuffle for Private Set Intersection. Three operations are performed at each stage: merge, compare, and sort.

Sort-Merge-Compare-Shuffle (SMCS). In scenarios where $m << |\Omega|$, more efficient solutions than Bitwise-AND can be devised. Here, we present one of the most complicated circuits in our benchmarks which is the generalization of the approach presented in [HEK12] from two-party

setting to any n-party case. As the input to this circuit, each set is represented as a vector of m integers where each integer is b-bit. We will first explain the solution for two sets only. The intersection of two sets can efficiently be computed using three operations: sort, merge, and compare. First, each of these two sets should be sorted. Then by merging the two sorted sets, all elements in common will be brought together. Finally, by comparing adjacent elements, one can find the common elements in both sets. Since the set intersection is an associative operation, one can express the set intersection of n sets as a consecutive set intersection of two sets until reaching the final result. Therefore, the SMCS circuit has a binary tree structure where at each node, the intersection of two sets are computed. The final node computes the final intersection of all sets. Note that the first sort operation can locally be computed by each participant since it is independent of the other parties' private data. A final shuffle operation is needed in order to eliminate the information leakage which we describe later in this section. Without loosing any generality, assume that the number of sets (participants) is a power of two. If this is not the case, dummy nodes can be avoided in the tree structure. Please see Figure 6.4 for a high-level description of the SMCS circuit.

We now elaborate on each part of the SMCS circuit. The challenge is that the merger and sorter circuits should have a *fixed* structure and *non-random access* to the intermediate values since random access is a very costly operation in the MPC protocols. We rely on the *bitonic* merger and sorter circuits that satisfy this condition. Bitonic sort is one of the sorting networks that is an efficient circuit-based realization of a sorting algorithm. Input numbers are given to the circuit and after series of conditional swap operations, a sorted list is given as the output of the circuit. The only operation used in the circuit is conditional swap: given two input numbers, swap them if they are not sorted and do not swap them otherwise. The bitonic sort has a recursive structure. It first sorts each half of the input and then merges the two sorted lists. The base case is a circuit that sorts only two numbers which is equivalent to a conditional swap module. Our implementation of the bitonic sort circuit is also a recursive hardware description code.

The second half of the bitonic sorter represents the *bitonic merger* circuit. The input to the bitonic merger must be a bitonic sequence. A sequence x_i of numbers is called bitonic if for some k ($0 \leq k < m$):

$$x_0 \leq x_1 \leq \ldots \leq x_k \geq \ldots \geq x_{m-1} \geq x_m$$

or a circular shift of such sequence. Therefore, before merging the two sorted lists, one needs to reverse order the second list such that the concatenation of two lists be a bitonic sequence. This reverse-ordering should take place for input sets as well as for intermediate sets. Note that the reversing the order of a set does not incur any computation or communication cost and is realized as changing the order of wires in the circuit.

The second layer in the SMCS circuit is the *comparison* layer. After the merger layer, all identical elements in both sets are now beside each other. An intuitive solution is to have a series of comparison blocks that compare every two adjacent elements. However, it has been shown that having a 3-input comparison block as follows is more efficient [HEK12]:

$$\text{CMP3}\ (x_1, x_2, x_3) = \begin{cases} x_2 \ \ if\ x_1 = x_2 \mid x_2 = x_3 \\ 0^b \ \ otherwise \end{cases}$$

Given an array of $2m$ elements, we only need $m-1$ CMP3 blocks and one CMP block (compared to $2m$ CMP blocks).

The output of the comparison layer is an array of m numbers consisting of 0^b and the elements in the intersection of two sets. Before proceeding to the next stage (and similar to the first stage), the array has to be sorted. Note that the intermediate sets should not be revealed to any party since some information about the private input sets will be learned by other parties. Therefore, in contrast to the first stage, the sets should be sorted inside the MPC protocol.

At the end of all stages, the final set should be shuffled prior to be revealed in plaintext to all parties. This step is necessary because the final set potentially has a sequence of 0^b between

two common elements. The position of zeros (0^b) reveal the distribution of elements that were not in the intersection and belong to one (or multiple parties) only.

The shuffling layer can be realized using Waksman permutation network [Wak68] which takes as input an array and shuffles them based on the *control bits*. One of the parties is required to provide these control bits as well. However, this task makes one of the parties to have more control in the secure computation. For example, a dishonest party that is selected to provide the control bits can simply put all of them as zero which makes the shuffle layer ineffective and he can learn some information. As a result, we devise another solution that is secure but does not require more input from any party. The solution is to simply sort the final list before revealing it in plaintext. This approach is secure since all of the 0^b elements are brought together. More precisely, in all of the scenarios that the common elements are fixed, the final sorted set remains the same and an adversary cannot distinguish different scenarios. The overall complexity of the SMCS circuit is $O\big(nm\lg^2 mb\big) = O\big(nm\lg^2 m\lg|\Omega|\big)$ (compare with Bitwise-AND circuit with complexity $O\big(n|\Omega|\big)$).

Modular Structure. One of the advantages of using a generic secure multi-party computation protocols such as BMR is its modular nature and flexibility. Unlike customized protocols, additional functionalities and computations can be augmented to the circuit seamlessly. For example, and auditing step can be added before releasing the final result: the intersection set is revealed if and only if the number of elements in common is less than a threshold. Such auditing steps are favorable especially when Ω is small and an adversary can easily put his input set as the universal set in which case, he clearly learns the intersection of all other sets. As another example, it is very straightforward to build other variants of PSI such as PSI-Cardinality which only outputs the size of the intersection and not the elements.

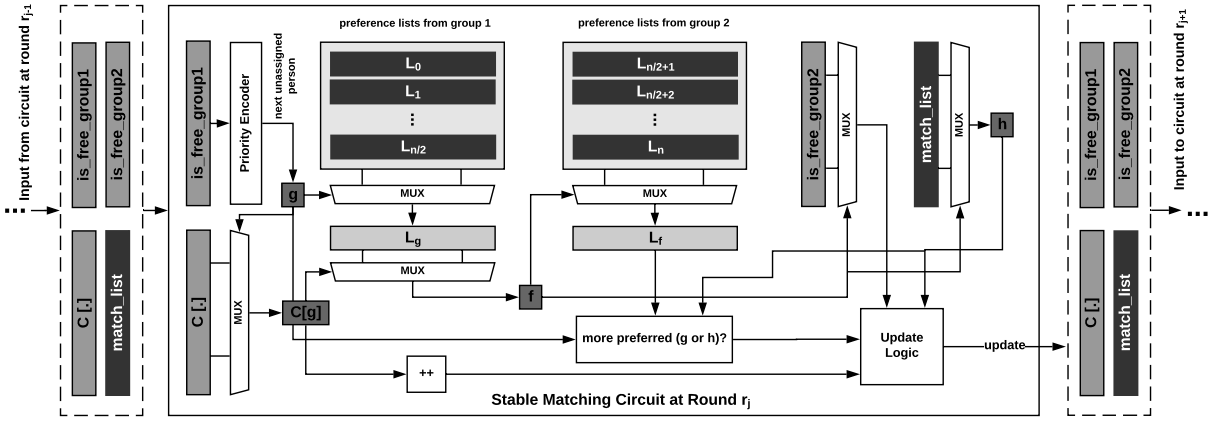


Figure 6.5: The circuit for stable matching unrolled for round r_j. The circuit takes as input the intermediate values from previous round r_{j-1}, processes the current round based on the preference lists, and outputs the updated values.

6.7 Stable Matching

Stable matching is the process of assigning the members of two groups to each other (one-to-one) where each person has a preference list. This assignment should satisfy the *stability* condition after the assignment: no two individuals should prefer to be matched with each other compared to their already assigned partners. In other words, the assignment is stable in a sense that no rematching will occur even if individuals are free to do so. Stable matching is one of the most complicated task in secure computation because of the complex and data-dependent memory accesses during the computation [RSS+17, DES16]. Accessing the memory when the address is a secret value is a very costly operation in secure computation since the actual value of the address should remain private. In order to realize this constraint, random access to the memory is implemented using multiplexers inside the garbled circuit. Hence, the address as well as the accessed data remain private since they are processed inside the garbled circuit.

In secure stable matching, the match list is computed while keeping the preference lists private to their respective owners. This problem has been studied in the recent literature [RSS+17, DES16] where the secure stable matching problem is reduced to a two-party secure computation

scenario. Each individual XOR-shares her preference list and sends it to two non-colluding servers who perform the secure computation. However, stable matching is inherently a *multi-party* problem and the assumption of two non-colluding servers may not be feasible in practice. To the best of our knowledge, we provide the *first* solution for multi-party secure stable matching.

Input: Party P_i holds a preference list L_i with size of m. Each list is an array of $\lg(\frac{n}{2})$-bit numbers (IDs of the other group's members) sorted from the most preferred to the least.

Output: The match list: an array of size $\frac{n}{2}$ where each number is $\lg(\frac{n}{2})$-bit.

Parameter(s): The size of the preference list m.

6.7.1 Circuit Design

Gale and Shapley [GS62] were first to formalize the stable matching problem and proposed an algorithm that can find the matching list. During the computation, the matching list stores the temporary assignment of individuals. The algorithm works as multiple rounds. In each round r_j, an unassigned individual from group 1, say g, is selected. The circuit identifies if this individual can be assigned to the most preferred person given the preference list of g. Note that in this algorithm, each element of the preference list is accessed only once. If the match is not accepted, next preferred ID is selected in future rounds. The number of attempts for each individual from group 1 is stored as an array of counters C, i.e., $C[i]$ denotes the number of attempts for individual i.

Assume that at round r_j, unassigned individual g from group 1 is selected. The circuit first accesses the C array and finds the next preferred individual that is not already processed, let's call that ID f, $f = L_g[C[g]]$. The circuit increases the number of attempts for g by incrementing $C[g]$. If f is unassigned, the circuit assigns f from group 2 to g from group 1. Otherwise, it is necessary to determine whether f prefers g or his already assigned person h from group 1.

The decision can be made by comparing the index of h and g in the preference list of f. That is, comparing $index\_of(h)$ and $index\_of(g)$ in L_f. If $index\_of(g)$ is less than $index\_of(h)$ in L_f, it means f prefers g over h. In this case, f is assigned to g and h gets unassigned and nothing happens otherwise.

In the next round, another unassigned individual from group 1 is selected and the same computation is performed. The process continues until all elements in the preference lists are processed. Note that this algorithm can be realized using a sequential circuit [RSS+17] but since there is currently no methodology for making the BMR protocol compatible with sequential circuits, we need to unroll the circuit as depicted in Figure 6.5. The overall complexity of the combinational circuit for secure stable matching is $O\big(n^3\ m\ lg(n)\big)$.

6.8 Nearest-Neighbor Search (NNS)

In this benchmark, each party holds an attribute value v_i and one of the parties (P^*) is interested to learn which party (or parties) has the most similar (closest) attribute to her, given a certain similarity metric. NNS has many applications in classification, data mining, recommender systems, and proximity search. A privacy-preserving solution enables a client to find the most similar profiles without revealing her attribute. For example, consider an online dating website where each person creates a profile containing sensitive information about his/her age, personal preferences, and the zipcode of where she/he lives. Today, a centralized server knows all of the clients' information and provides the match result to each party. This computation model is also prone to internal and external attacks where clients' sensitive information is revealed to the attacker. In contrast, we propose a privacy-preserving method that is decentralized and is provably secure even if all other parties collude. Our methodology is modular and can be realized for any similarity metric since only the comparison module has to be modified.

Input: Party P_i holds an attribute value v_i.

Output: The set of k_n closest (most similar) attributes to party P^* input (v^*) given a similarity function $sim(.,.)$.

Parameter(s): Number of bits b required to represent each attribute. Number of nearest neighbors to be found (k_n).

6.8.1 Circuit Design

Our implementation for NNS is the generalization of the combinational circuit in [SHSK15] that computes 1-nearest neighbor search. We generalize the combinational circuit for any value of k_n. The core block of the design is a module that takes as input k_n distance values along with a new distance value and outputs the corresponding k_n minimum distances from the total $k_n + 1$ inputs. This module is instantiated n times where each instance processes the input from one party. Please note that in contrast to the two-party garbled circuit protocol, there is no known solution to use the sequential circuit in the general multi-party variant. Therefore, sequential circuits provided for k-nearest neighbors cannot be used. The overall complexity of the combinational circuit for NNS is $O(n\ k_n\ b)$. `MPCircuits` supports the distance computation module to be any distance metric, e.g., the Hamming distance, euclidean distance, edit distance, or taxicab distance. In the following, the reported experimental results correspond to the Hamming distance.

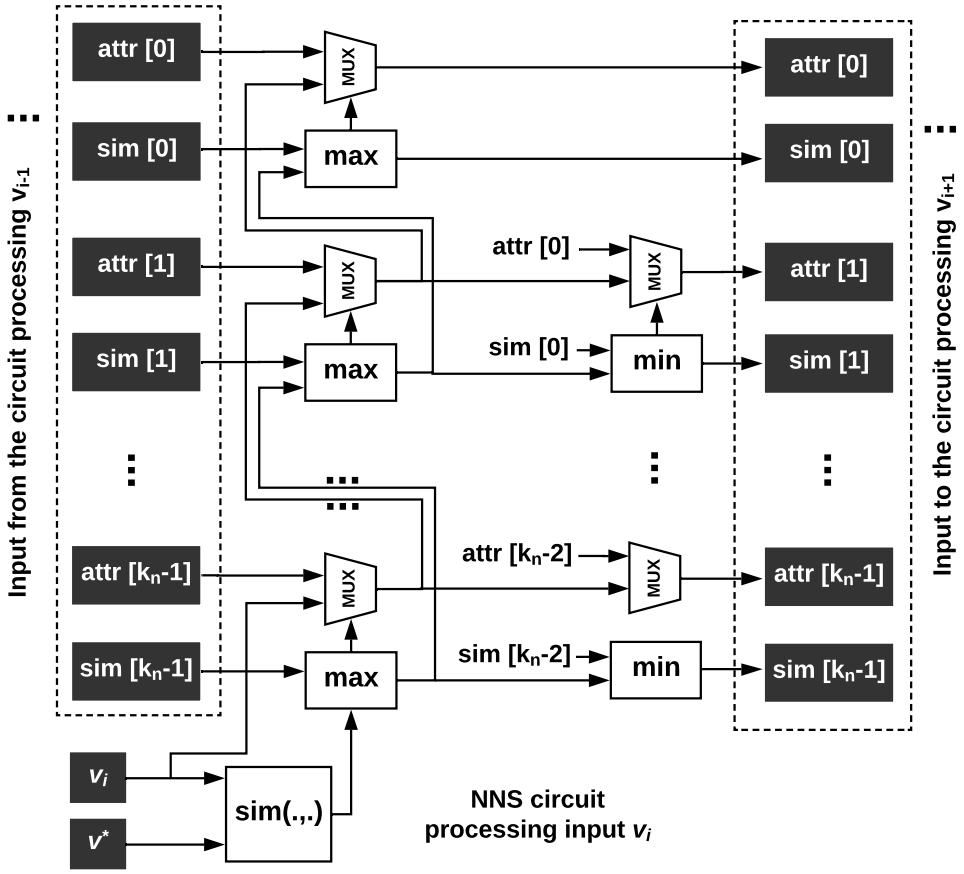


Figure 6.6: The circuit of the Nearest-Neighbors Search (NNS) that finds the k_n most similar attributes to v^*. In the experimental results, the circuit is unrolled for n times.

6.9 Experimental Results and Related Work

We first discuss the metrics by which we characterize each application. We outline the metrics and the reason for their importance in practical realization of the MPC protocols.

- Execution time (T): The total execution time of the protocol comprises the time required for garbling/evaluating the circuit (T_{GE}) as well as time spent on the communication T_C. In a general case, these two can overlap in time depending on whether the implementation is pipelined/multi-threaded or not and hence, $T \leq T_{GE} + T_C$. The distinction between the two timing parameters is important since T_{GE} mostly depends on the computational power, whereas, T_C depends on the network quality (delay and bandwidth).

- Communication (*Comm*): Maximum number of bytes exchanged between any two parties. The "maximum" is required for protocols in which communication between parties are asymmetric. In the BMR protocol, the communication between each two parties can be computed as the multiplication of number of non-`XOR` gates, a constant factor (=9), number of parties minus one ($n-1$), and the bit-length of each wire label (usually 128).

- Memory footprint and scalability (*Mem*): One of the important characteristics for each MPC protocol is the amount of memory allocated in the end-to-end execution. Protocols/frameworks that consume a high volume of memory have limited scalability in real-world scenarios where the input size from each party is large.

6.9.1 Experimental Setup

The experiments are performed on a server equipped with 24 core Intel(R) Xeon(R) E5-2650 v4 @2.20GHz CPU with 256GB of RAM. We run all n parties in the same LAN network with 20ms round-trip latency and 10Gbps bandwidth. Synopsys Design Compiler 2015.06-SP2 is used to synthesize the Boolean circuits. *RC* is constant in all of our benchmarks for different values of parameters since our prototype implementation is based on [BELO16] which has a constant round complexity. The SCAPI library utilizes Advanced Encryption Standard (AES) encryption and naturally benefits from the AES-NI which is supported by our machine. In our experimental results, we have used built-in Ubuntu `time` tool with `-f '%M'` flag to determine the memory footprint.

6.9.2 Auction

We perform experiments for different numbers of participants (n) in the auction for two values of b. Table 6.1 shows the results. As can be seen, the optimized Boolean circuits using `MPCircuits` technology libraries reduce the number of `AND` gates by 3.3×. Bogetoft et

al. [BDJ+06] have proposed a solution for secure auction. Their solution is based on multiple "Trusted Third Parties (TTPs)". TTPs compute the true outcome of the auction on behalf of the bidders. In this computation model, if all TTPs collude, the real input of all parties are revealed, whereas, in our approach, all parties securely process the auction and even if all other parties collude, nothing is revealed. The approach of [Hua16] also requires a separate party called "Auction Issuer". The methodology in [MNPS04] additionally requires outsourcing the computation to two TTPs. Larson et al. [LHL+15] design a method based on a verifiable secret sharing scheme. The drawback of their approach is that not all participants in the auction are involved in the secure computation protocol and the security relies on the evaluators. Therefore, our solution is the *only* solution that (i) has constant round complexity and (ii) guarantees security even for cases where all other parties are corrupted.

Table 6.1: Secure Auction.

		Non-optimized		**Optimized**						
b	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
16	4	69	324	261	97	0.74	0.62	2.39	0.04	10.25
	8	140	761	600	228	1.69	1.91	6.62	0.22	10.29
	16	281	1638	1281	492	3.51	4.48	15.06	1.01	18.14
32	4	133	660	534	194	0.74	0.66	3.41	0.08	10.31
	8	269	1547	1229	454	1.66	1.83	6.50	0.44	10.36
	16	539	3324	2621	975	3.48	4.34	16.85	2.01	30.65

6.9.3 Voting

Table 6.2 shows the experimental results for different number of parties (voters) and candidates. As can be seen, `MPCircuits` is between 1.4-2.7× more efficient compared to standard utilization of logic synthesis tools. Civitas [CCM08] is a secure voting system which is verifiable and coercion-resistant but requires five different type of agents for its execution. Fujioka et al. [FOO92] also propose a solution for secure auctions but it requires two additional

entities called administrator and the counter conspire. In contrast, our solution does not involve any additional agents or entities.

Table 6.2: Secure Voting.

		Non-optimized		**Optimized**						
n_c	n	#XOR	#AND	#XOR	#AND	*OT* (s)	T_{GE} (s)	*T* (s)	*Comm* (MB)	*Mem* (MB)
2	8	7	17	18	8	1.57	1.77	9.35	3.3 KB	10.09
	16	19	43	45	16	3.29	4.29	13.76	0.02	10.09
4	4	17	50	23	37	0.71	0.54	3.25	0.02	10.09
	8	49	128	105	79	1.64	1.80	6.46	0.08	10.08
	16	123	294	249	147	2.99	4.11	14.23	0.30	10.08
8	16	250	739	545	388	3.40	4.01	15.40	0.80	15.30

6.9.4 Private Set Intersection

Table 6.3 shows the experimental results of Bitwise-AND circuit for different sizes of the universal set and different numbers of parties. For all PSI experiments, parameter *m* is set to 16. The corresponding results for the SMCS circuit are shown in Table 6.4. As can be seen, the optimized Boolean circuits using `MPCircuits` technology libraries reduce the number of `AND` gates by 4.2×.

There has been an extensive research focus on the Private Set Intersection (PSI) problem for a two-party situation [DCT10, PSSZ15, HEK12]. In [HEK12], authors propose a method for two-party PSI based on garbled-circuit approach. To the best of our knowledge, the only solution that is proposed for secure multi-party private set intersection is a recent work by Kolesnikov et al. [KMP+17]. Their approach is a customized solution that is optimized only to perform PSI in an identical security model as this work. Their computation platform is comparable but more powerful than ours. In the LAN setting, for a set size of 2^{16} and 10 parties, their total running time is 12 seconds with 23MB of communication. Whereas, for a universal set of size 10^5 ($\sim 2^{17}$) and

8 number of parties, our running time is 24 seconds with 314MB of communication. Although our solution is less optimized, we want to emphasize that we have proposed a generic solution to create any functionality, whereas, their solution is specially optimized for PSI. In addition, our solution has a very modular structure and can easily be modified to support other variants of the PSI, e.g., PSI cardinality in which only the *number* of mutual elements is revealed. Moreover, in Bitwise-AND circuit, the actual size of each party's set is not revealed since the inputs are fixed-length binary vectors.

Table 6.3: Private set intersection (Bitwise-AND variant).

		Non-optimized		**Optimized**						
$\|\Omega\|$	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
10^4	4	0	3.00E+04	0	3.00E+04	0.94	0.69	3.89	12.36	65.24
	8	0	7.00E+04	0	7.00E+04	2.73	1.99	9.46	67.29	403.94
	16	0	1.50E+05	0	1.50E+05	12.61	4.74	30.46	308.99	2835.59
10^5	4	0	3.00E+05	0	3.00E+05	1.99	0.88	6.80	123.60	584.44
	8	0	7.00E+05	0	7.00E+05	11.82	2.89	24.05	672.91	3892.61

Table 6.4: Private set intersection (SMCS variant).

		Non-optimized		**Optimized**						
b	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
16	4	1.05E+04	7.77E+04	5.02E+04	1.86E+04	0.82	0.56	3.52	7.66	52.57
	8	2.42E+04	1.81E+05	1.16E+05	4.30E+04	2.42	1.76	6.59	41.37	280.42
	16	5.15E+04	3.88E+05	2.48E+05	9.19E+04	9.65	4.40	41.50	189.34	1843.72

6.9.5 Stable Matching

Table 6.5 shows the circuit size as well as the experimental results for different group sizes and preference list lengths. As can be seen, the optimized Boolean circuits generated by `MPCircuits` technology libraries have 1.6-2.4$\times$ lower number of `AND` gates. To the best of our knowledge, there has been no prior solution for multi-party secure stable matching. State-of-the-

art solutions reduce the task to two-party secure computation problem [RSS+17, DES16]. All parties outsource the computation to two servers which are assumed to not collude. While these solutions can scale to bigger set sizes, they rely on additional servers to find the match list on their behalf. If two servers collude, they can learn the preference list of all individuals in plaintext. In contrast, our security model is much stronger where *any* number of corrupted parties cannot learn the preference list of other individuals and the solution does not require additional servers for the computation.

Table 6.5: Secure stable matching.

		Non-optimized		**Optimized**						
m	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
2	8	1.83E+02	1.11E+03	7.28E+02	5.35E+02	1.35	1.54	6.96	0.51	10.42
4		7.51E+02	4.55E+03	2.95E+03	2.24E+03	1.45	1.54	7.05	2.16	21.39
3	12	1.69E+03	9.61E+03	5.21E+03	6.03E+03	2.35	2.52	10.94	9.11	84.50
6		5.48E+03	3.10E+04	1.74E+04	1.90E+04	2.94	2.57	11.83	28.63	230.88
4	16	4.22E+03	3.70E+04	2.38E+04	1.67E+04	4.13	3.74	15.35	34.32	341.76
8		1.18E+04	1.11E+05	7.32E+04	4.66E+04	5.97	4.04	18.23	95.95	920.10

6.9.6 Nearest-Neighbor Search

Due to the space limitation, we report the results for $b = 32$ in Table 6.6. The distance function is Hamming Distance (HD). However, the circuit can be instantiated for any value of b. As can be seen, `MPCircuits` customized libraries result in 3-3.2$\times$ performance improvement compared to standard utilization of logic synthesis tools. Songhori et al. [SHSK15] propose a solution based on Garbled Circuits [Yao86b]. However, their approach is limited to the two-party setting only. Similarly, Chen et al. propose a methodology based on homomorphic encryption, garbled circuits, oblivious RAM for the two-party scenario [CCD+19, CCD+18]. Qi et al. [QA08] create a scheme based on Homomorphic encryption for two-party settings. Perhaps the most similar work to ours is [SKK09] where they support a multi-party setting. Nevertheless, they

have not implemented their scheme.

Table 6.6: Secure k-nearest neighbor search.

		Non-optimized		Optimized						
k_n	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
1	8	7.64E+02	1.77E+03	1.79E+03	5.56E+02	1.61	1.77	7.40	0.53	10.75
	16	1.50E+03	3.68E+03	3.73E+03	1.16E+03	3.26	3.76	16.07	2.39	38.27
2	8	8.66E+02	3.31E+03	2.73E+03	1.08E+03	3.26	1.78	7.40	1.04	16.06
	16	1.67E+03	7.25E+03	5.82E+03	2.37E+03	3.48	1.79	16.32	4.88	66.54
3	8	9.77E+02	4.64E+03	3.62E+03	1.52E+03	1.59	1.70	9.26	1.46	20.42
	16	1.85E+03	1.06E+04	8.20E+03	3.50E+03	3.52	4.01	14.85	7.21	98.21

6.9.7 Scaling Up Circuit Generation

In our experiments, we observe that the main bottleneck for running the BMR protocol for any number of parties higher than 16, is scaling the number of communication ports as well as the number of physical machines. In BMR, each party needs to communicate with all other parties using a secure communication channel, resulting in a total of $\frac{n\ (n-1)}{2}$ point-to-point communication. Nevertheless, we want to emphasize that our approach for generating the optimized circuits can scale up to considerably higher sizes. For example, Table 6.7 and Table 6.8 show the number of Boolean gates for higher parameter sizes in secure auction and secure NNS, respectively. As can be seen, the circuit generation can easily scale up to 512 and 1024 number of parties.

Table 6.7: Circuit generation for higher number of participants in secure auction.

b	n	#XOR	#AND	b	n	#XOR	#AND
16	256	2.08E+04	8.42E+03	32	256	4.25E+04	1.66E+04
	512	4.18E+04	1.69E+04		512	8.51E+04	3.33E+04
	1024	8.36E+04	3.38E+04		1024	1.70E+05	6.67E+04

Table 6.8: Circuit generation for higher number of participants in the secure k_n-NNS search when $b = 32$.

k_n	n	#XOR	#AND	k_n	n	#XOR	#AND
	128	2.04E+04	4.88E+04		128	3.13E+04	7.20E+04
2	256	4.10E+04	9.82E+04	3	256	6.30E+04	1.45E+05
	512	8.22E+04	1.95E+05		512	1.27E+05	2.89E+05

6.10 Summary

We present `MPCircuits`, the first automated methodology to generate optimized Boolean circuits for secure multi-party computation (MPC). The Boolean circuit generation is a key step to employing the MPC protocols. We leverage industrial logic synthesis tools and transform the problem of generating optimized circuits for MPC to a logic synthesis problem. Our solution is modular and generic and can be adopted by different MPC protocols and implementations. To illustrate the practicality of our approach, we design and implement Boolean circuits for five compelling tasks in MPC. Namely, we consider auction, voting, private set intersection, stable matching, and nearest neighbor search. We perform extensive experimental evaluation of all five benchmarks based on the Beaver-Micali-Rogaway (BMR) protocol and show that `MPCircuits` automatically generates optimized circuits that require up to 4.2$\times$ less garbled gates.

6.11 Acknowledgements

This chapter, in part, has been published at the Proceedings of 2019 IEEE Hardware Oriented Security and Trust (HOST) and appeared as: M. Sadegh Riazi, Mojan Javaheripi, Siam U Hussain, Farinaz Koushanfar, "MPCircuits: Optimized Circuit Generation for Secure Multi-Party Computation" [RJHK19]. The dissertation author was the primary author of this material.

Chapter 7

PriSearch: Privacy-Preserving Text Search

We propose PriSearch, a provably secure methodology for two-party string search. The scenario involves two parties, Alice (holding a query string) and Bob (holding a text), who wish to perform a string search while keeping both the query and the text private without relying on any third party. Such privacy-preserving string search avoids any data leakage when handling sensitive information, e.g., genomic data. PriSearch provides an efficient solution where two parties only need to interact for a constant number of rounds independent of the query and text size. Our approach is based on the provably secure Yao's Garbled Circuit (GC) protocol that requires the string search algorithm to be described as a Boolean circuit. We leverage logic synthesis tools to generate an optimized Boolean circuit for PriSearch such that it incurs the minimum communication/computation cost. We achieve approximately $2\times$ and $140\times$ performance improvements compared to the best prior non-GC and GC-based solutions, respectively.

7.1 Introduction

Privacy-preserving string search allows one party, i.e., the query holder, to learn whether her text query is present in a text document held by another party while keeping both the text and

the query private to their respective owners. The prior work on string search typically aims to reduce execution time or memory usage, and less focus has given to security and data privacy. A number of obfuscation and anonymization approaches have been proposed, but it has been shown that they have serious pitfalls [GMG+13, HSR+08]. The ultimate solution is to keep the query and the text private to their respective owners. Such *privacy-preserving* string search allows two parties, Alice holding the query and Bob holding the text, to perform string search while keeping both the query and text private.

There have been a few attempts to solve the privacy-preserving string search [BEDM+12, Ver11, Fri09]. The shortcomings of the previous work can be categorized as follows: (i) Lack of solid security proofs for heuristic and custom designed protocols [GMG+13, HSR+08]. (ii) The number of communication rounds increases as the sizes of text and query increase [WCS15]. (iii) Prohibitive computation overhead, e.g., solutions based on Homomorphic Encryption (HE) require several expensive asymmetric encryptions [DCFT13]. (iv) Extension to support other variants of the string search, e.g., providing the number of matches and longest prefix match is not trivial [Ver11].

PriSearch provides an efficient solution for privacy-preserving string search based on Yao's Garbled Circuit (GC) protocol. GC is a provably secure protocol that allows two parties to evaluate a function on their private inputs. A challenging step in the GC protocol is to convert the underlying function (here the string search algorithm) into a Boolean circuit such that it incurs minimum communication/computation cost. Utilizing the GC protocol guarantees the privacy requirements, however, a naive implementation of string search as a Boolean circuit results in a huge cost because it requires multiple random accesses to the text (see Section 7.2.2). Here, the address of a random access depends on the query and the text, thus, it has to be hidden from both parties, a costly operation in GC which is called *oblivious access*. A *single* oblivious access to the text has linear computation and communication complexity $O(n)$ with respect to the text size (n). The cost of oblivious access can be improved using Oblivious RAM (ORAM) inside GC to

achieve polylogarithmic complexity $O(polylog\ n)$ per access. Sadly, the ORAM scheme with the best asymptotic complexity [WCS15] needs at least $O(\log n)$ sequentially dependent rounds of communication per access, making the overall execution prohibitively slow.

We introduce PriSearch, a novel privacy-preserving string search which does not need random access to the text and it has constant round complexity ($O(1)$). The computations required in PriSearch are mainly based on symmetric key encryption and are far less expensive compared to HE. We also design new synthesis libraries and leverage logic synthesis tools to generate an optimized sequential circuit for PriSearch automatically to achieve minimal cost in GC. Moreover, PriSearch is designed such that it can be easily extended to support multiple variants of string search. In brief, our contributions are as follows:

- Analyzing five major string search algorithms and their performance in the GC protocol and providing their respective complexity.
- Developing a string search algorithm (based on the Knuth-Morris-Pratt algorithm) that does not require random access to the text.
- Automatically generating the optimized Boolean circuit for string search by leveraging logic synthesis tool and new synthesis libraries.
- Performing extensive experiments on six different benchmarks and achieving approximately $2\times$ and $140\times$ performance improvements compared to the best prior non-GC and GC-based solutions, respectively.

7.2 PriSearch

In the GC protocol, the underlying function that is evaluated securely has to be described as a Boolean circuit. In PriSearch, the function is string search and the inputs are the text and the query. There are several algorithms for string search and each one incurs a different cost if

described as a Boolean circuit for the GC protocol. In the following, we explore these algorithms and in Section 7.2.2, their computation/communication costs in GC are discussed.

7.2.1 String Search Algorithms

We briefly describe and compare brute-force and four most efficient string search algorithms [SW11].

- **Brute-force**: The simplest algorithm is to start from the beginning of the text and compare it with the query. If the first characters match, proceed to the next ones. If all characters have been matched, then the algorithm returns true. Upon a mismatch, the starting pointer of the text is incremented by one and the search is continued until the end of the text. This algorithm has $O(mn)$ computational complexity.

- **Rabin-Karp**: This algorithm compares the hash value of the query against the hash value of each part of text of size m. The elementary implementation incurs complexity of $O(mn)$ but using a more efficient hash computation, called Horner's method, the complexity can be reduced to $O(n)$.

- **KMP**: Knuth-Morris-Pratt algorithm is similar to brute-force algorithm. However, upon a mismatch, the pointer on the text is not incremented by one and instead, it jumps for a certain value based on a precomputed information. The complexity of KMP is $O(n)$.

- **Boyer-Moore**: This algorithm starts the comparison from the *end* of the query. If it matches, it compares the second to the last character and so on. If a mismatch happens, it chooses the next character for comparison in the text based on the mismatched character. On average, it takes $O(n/m)$ to finish and is considered one of the best algorithms in the literature but in the worst-case scenario, it takes $O(nm)$.

- **FSM-based**: The core idea is to create a Finite State Machine (FSM) based on the query and feed the FSM with one character of the text at a time. Based on the current state of the FSM (number of matches so far) and the new input character of the text, the next state is chosen. If it reaches the final state, a match is found. Its computational complexity is $O(n)$.

Precomputed information. Beside the computational complexity, the other difference between these algorithms is the size of the precomputed information that they need. In the GC protocol, precomputed data has to be accessed without revealing any information about the text or query. This access cost increases with the size of the precomputed information. Here, we analyze the size of precomputed information for all of the algorithms.

Brute-force algorithm does not require any precomputed data. Rabin-Karp requires $O(1)$ precomputed information, only a hash value is computed and stored. In KMP algorithm, once a mismatch is identified, the size of the jump on the text is determined based on the mismatch location in the query (m possible locations), whereas, in the Booyer-Moore algorithm, it is based on the mismatched character in the text (R possible characters). Therefore, KMP needs to store $O(m)$ precomputed information while Booyer-Moore needs to store $O(R)$.

FSM-based algorithm makes the decision based on the current state (m possible states) and the input character of the text (R possible characters). Thus, it must store mR possible combinations ($O(mR)$ precomputed information).

7.2.2 String Search in GC

We have identified three main requirements for efficient realization of string search in the GC protocol.

1. Non-random access to the text. In Booyer-Moore and KMP algorithms, the access sequence of the text depends on the content of the text and the query. Therefore, the access pattern should be hidden in order to keep the text and the query private. This can be done in the GC

Table 7.1: Summary of characteristics for different string search algorithms.

Algorithm	Non-random access to the text	Low-cost operations	Linear-size pre-computed information
Rabin-Karp	✓	-	✓
KMP	-	✓	✓
Booyer-Moore	-	✓	✓
FSM-based	✓	✓	-
PriSearch	✓	✓	✓

protocol using a random access memory (multiplexers and flip-flops) in the Boolean circuit. The computation/communication cost of naive implementation of random access memory inside GC is $O(n)$ per access. Therefore, complexities of Booyer-Moore and KMP algorithms in GC are $O(n^2 m)$ and $O(n^2)$, respectively.

A better solution for random access memory in GC is by utilizing Oblivious Random Access Memory (ORAM). The ORAM scheme with the best asymptotic complexity incurs $O(polylog(n))$ access cost [WCS15]. However, each access to ORAM requires $O(\log n)$ sequentially dependent rounds of communication between two parties. Considering the network latency, ORAM makes the overall private search extremely slow and limits its scalability for large n. In contrast, the access sequence to the text in Rabin-Karp and FSM-based algorithms do not depend on the content of the text nor the query. In these algorithms, each character of the text is accessed strictly in order and only once. As such, the costly random access memory is not required. Therefore, the most efficient string search algorithm for GC has to have a non-random access pattern to the text.

2. Low-cost operations. The big O notation expresses how the private search protocol scales as n and m grow. However, the constant coefficient hidden in the Big O notation plays an important role in the overall performance. The constant coefficient corresponds to the computation that needs to be performed for processing one character of the text. In the sequential GC, the constant coefficient is proportional to the size of the Boolean circuit implemented for the string

search algorithm. Rabin-Karp algorithm includes modular exponentiations and multiplications that result in a very large circuit size. In contrast, Our objective is to construct the smallest possible circuit.

3. Linear-size precomputed information. The FSM-based algorithm meets the first two requirements. However, a precomputed information of size $O(mR)$ has to be accessed once for processing each character of the text, yielding the $O(nmR)$ complexity. Our goal is to have the minimum precomputed data possible to reduce the overall complexity. Rabin-Karp is the only algorithm that needs $O(1)$ precomputed information. In the Rabin-Karp algorithm, once the hash value of a part of the text becomes equal to the hash value of the query, a final check should take place to verify a true match between the two strings. In GC, one cannot distinguish whether a hash match occurred or not, hence, the final check has to be performed in every iteration. This makes Rabin-Karp algorithm's performance in GC even worse than the brute-force algorithm.

None of the above algorithms has $O(1)$ precomputed information that also satisfies the first two requirements. Thus, our goal is to have an algorithm with linear size precomputed information in terms of m (or R). Table 7.1 summarizes the characteristics of different string search algorithms.

Table 7.2: Summary of different algorithms for string search and their complexity; $R = |\Sigma|$, n is the number of characters in the text, and m is the number of characters in the query. The missing complexities in the fifth column means utilizing ORAM does not improve the performance.

Algorithm	Plaintext Average-case Complexity	Plaintext Worst-case Complexity	GC Complexity ($O(1)$ Round Complexity)	GC + ORAM Complexity ($O(n \log n)$ Round Complexity)	Size of Precomputed Information
Brute-force	$O(nm)$	$O(nm)$	$O(nm \log R)$	-	-
Rabin-Karp	$O(n)$	$O(nm)$	$O(nm \log R)$	-	$O(1)$
KMP	$O(n)$	$O(n)$	$O(n(n+m+\log R))$	$O(n(\log^3 n + \log^3 m + \log R))$	$O(m)$
Booyer-Moore	$O(n/m)$	$O(nm)$	$O(nm(n+m+R+\log R))$	$O(nm(\log^3 n + \log^3 m + \log^3 R + \log R))$	$O(R)$
FSM-based	$O(n)$	$O(n)$	$O(nmR)$	$O(n(\log^3 mR))$	$O(mR)$
PriSearch	$O(n)$	$O(n)$	$O(nm \log R)$	-	$O(m)$

7.2.3 PriSearch Algorithm

We design PriSearch, a string search algorithm, based on KMP that satisfies the three aforementioned requirements for efficient privacy-preserving string search. Algorithm 8 shows the pseudocode of the PriSearch algorithm. The algorithm is implemented as a sequential Boolean circuit in order to be evaluated in the GC protocol. Each iteration of the algorithm is computed in one clock cycle of the sequential circuit. As shown in the Algorithm 8, at each clock cycle, a certain character of the text ($T[l]$) is accessed. The index of this character, l, increases by one regardless of the content of the query and the state of the matching process at each iteration (Requirement 1). The index i shows which character of the query Q is compared against $T[l]$ at each iteration. PriSearch algorithm starts by comparing the first character of the text ($l = 0$) with the first character of the query ($i = 0$). If they match, i is incremented by one. If they do not match, similar to KMP algorithm, the algorithm uses a precomputed array P in order to update i for the next iteration. The algorithm finishes when l reaches the n.

Figure 7.1 shows an example to illustrate the steps in PriSearch. The final position of the matched substring is shown with a solid green line. After the first four characters (shown in blue color) are matched, a mismatch happens (shown in red). At this moment, it is not possible to reset i to zero and l to one, similar to the brute-force algorithm, because l is automatically increased by one. This means that one cannot access $T[1:3]$ to restart the search process. However, one can leverage the knowledge that $T[0:3]$ is equal to $Q[0:3]$ to properly update i.

Array P stores the index of the next character to compare (new i) based on the current mismatch position (i). Since the mismatch occurs at $i = 4$ and given that $P[4] = 2$, $T[l]$ has to be compared against $Q[2]$, i.e., updating i to 2. Thereby, $T[l] = T[4] =$‘A’ is compared with $Q[P[4]] = Q[2] =$‘A’. This comparison has to be done in the current iteration because $T[4]$ cannot be accessed in the next iteration (unlike KMP algorithm). Since $T[4]$ and $Q[2]$ are equal, in the next iteration $T[5]$ is compared with $Q[3]$ ($i = P[i] + 1 = 3$). $Q[3:4]$ also match in the following iterations and the output o becomes 1. The only operations used in PriSearch algorithm are

Algorithm 8 PriSearch.

Inputs: Bob's text T, Alice's precomputed array P and query Q.
Output: Output o indicating match has been found or not.

```
1:  i = 0
2:  o = 0
3:  // processing one character of the text per clock cycle
4:  for l = 0 to n − 1 do
5:      c = T[l]
6:      if Q[i] == c then
7:          // next character match
8:          i = i + 1
9:          if i == m then
10:             // complete string match is found
11:             o = 1
12:         end if
13:     else if Q[P[i]] == c then
14:         // mismatch, check the array P to see where
15:         // to check next
16:         i = P[i] + 1
17:     else
18:         // total mismatch, restart from beginning of Q
19:         i = 0
20:     end if
21: end for
```

the comparison of two characters and addition by one which are far less expensive operations than modular exponentiations and multiplications (Requirement 2). We design a new synthesis libraries that include optimized implementations of these operations (i.e., addition, comparison, and selection) for the GC protocol.

Array P is precomputed based on query Q using Algorithm 9, also used in KMP. $P[i]$ is used when query has been matched for the first i characters but not for the $(i+1)^{th}$ character. Upon a mismatch, the search needs to be restarted from $T[l-i+1]$. However, instead of $T[l-i+1:l-1]$, one can use $Q[1:i-1]$ to search against the query itself since $Q[0:i-1]$ is equal to $T[l-i:l-1]$. The content of array P is independent of the text and only depends on whether or not any prefix of the query is repeated within itself. In PriSearch, Algorithm 9 is done offline and locally by Alice, the owner of the query. Therefore, precomputation is performed

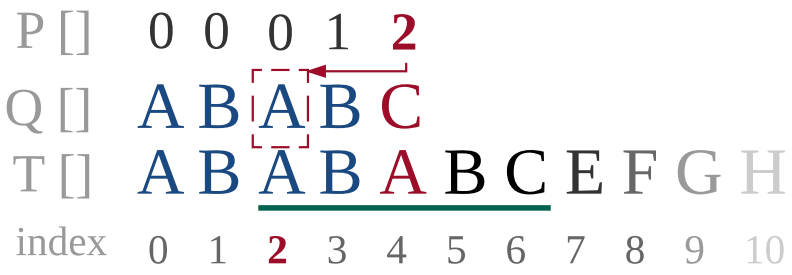


Figure 7.1: Illustration of steps in PriSearch algorithm.

without engaging in the GC protocol and it takes several orders of magnitude less time than the main string search. Along with the query Q, Alice provides the precomputed array P of size m (Requirement 3) as her secret inputs to the GC protocol.

Table 7.2 summarizes the complexity of different string search algorithms in plaintext (average-case and worst-case) and in GC. Plaintext complexity is the computational complexity of the algorithm when it runs locally without the GC protocol. GC complexity provides computation/communication complexity when an algorithm is described as a Boolean circuit and garbled/evaluated in the GC protocol. The difference between plaintext and GC complexities arises from two reasons: (i) In plaintext, for a given conditional (`if`) statement in the algorithm, only the taken branch of the statement is evaluated. In contrast, in the GC protocol, both branches should be garbled/evaluated in order to avoid information leakage. (ii) Random access to an array of size n has an $O(1)$ complexity in plaintext, whereas in the GC protocol, as mentioned before, it has $O(n)$ when using naive memory (GC complexity column) and $O(polylog(n))$ when using state-of-the-art ORAM [WCS15] (GC + ORAM complexity column). However, ORAM needs $O(\log n)$ rounds of communication for a single memory access. This means the overall round complexity using ORAM is $O(n \log n)$, which considering the network latency, makes the overall string search prohibitively slow. On the contrary, the GC protocol without ORAM has a constant round complexity of $O(1)$. As can be seen, our algorithm achieves the best GC complexity with a constant round complexity.

The Boolean circuit of PriSearch is automatically generated using a standard synthesis

Algorithm 9 Precomputing array P, done by Alice.

Input: Query Q.
Output: Array P.

```
1:  P[0] = 0
2:  if length(Q) > 1 then
3:      P[1] = 0
4:  end if
5:  pos = 2 //current position filling in P
6:  ind = 0 //zero-based index in Q
7:  while pos < length(Q) do
8:      if Q[pos − 1] == Q[ind] then
9:          //case 1: substring continues
10:         P[pos] = ind + 1
11:         ind = ind + 1
12:         pos = pos + 1
13:     else if ind > 0 then
14:         //case 2: it doesn't match, fall back
15:         ind = P[ind]
16:         P[pos] = 0
17:     else
18:         //case 3: run out of candidates, ind = 0
19:         P[pos] = 0
20:         pos = pos + 1
21:     end if
22: end while
```

tool. The synthesizing constraints are set such that the number of AND gates in the circuit becomes minimum because according to the Free XOR optimization [KS08b], XOR gates do not incur any cost.

7.2.4 Different Variants of PriSearch

Up to this point, we have focused on the simplest form of the protocol, which is string search with 1-bit output, indicating whether the query is present in the text or not. We now show how PriSearch can be readily adapted to yield other variants of privacy-preserving string search. We briefly mention two of these variants together with the required changes to the main algorithm:

- **Providing the total number of matches**: One can keep track of the total number of matches using a counter. To do this, *counter* needs to be initialized and the following line has to be inserted after Line 10 in Algorithm 8:

  11: *counter* = *counter* + 1

- **Size of longest prefix match**: A very important and useful information in the string search is the length of the longest prefix of query matched in the text. For example, having "ABCD" as query and text as "ABCEFG", the algorithm should output 3. Although an exact match is not found, a prefix of size 3 ("ABC") is present in the text. The change required for this is to have another value, say *prefix*, and insert the following lines after Line 8:

  09: **if** *i ¿ prefix* **then**

  10: *prefix* = *i*

  11: **end if**

7.2.5 Security of PriSearch

The GC protocol is proven to be secure against honest-but-curious adversaries for any function $f(a,b)$ [BHKR13]. In our setting, the function is the PriSearch algorithm, a is Alice's input which is composed of precomputed array P and query Q, and b is Bob's input which is the text T. Thus, the security of PriSearch immediately follows from the security of the GC protocol.

7.3 Results

We perform our experiments using two Intel Core i7-2600 CPU @ 3.4GHz processors, 12GB RAM and 64-bit Ubuntu 14 operating system. The security parameter in our setup is 128-bit ($k = 128$). We describe the PriSearch algorithm in Verilog, a hardware description language. We then use Synopsys Design Compiler 2010 along with our constraints and synthesis libraries to generate its Boolean circuit.

Table 7.3 shows the results for different string search benchmarks and their comparison with previous work whenever applicable. PriSearch achieves 2× improvement compared to the best prior privacy-preserving string search [BEDM$^+$12] that is secure against the same adversary model. They report DNA pattern matching with a query of size 100 characters against a text with 100k characters takes 64 seconds [BEDM$^+$12] while PriSearch finishes the same task under 34 seconds. PriSearch can be used for real-time processing stream of characters. It can securely process up to 15k characters per second when searching for a query of size 10. Comparing our solution with the best GC implementations (utilizing KMP algorithm and state-of-the-art ORAM implementation [WCS15]), we observe more than 140× faster execution when $m = 100$, $n = 64k$, and $R = 256$.

Table 7.3: Timing and communication results for different benchmarks.

Benchmark	n	m	$\log_2(R)$	Precomp.(ms)	Online Comp.(s)	Comm.	Exec.(s)	Prior Art(s)	Improv.
DNA Matching 1 [Fri09]	10†	10k	2 bit	1.38	0.15	55.34MB	0.53	8	15×
DNA Matching 2 [BEDM$^+$12]	100k	100	2 bit	0.03	9.30	3.50GB	33.84	64	1.9×
Document Search	64k	100	8 bit	0.03	12.51	4.74GB	45.49	6424‡	141×
Longest Prefix Search	100k	10	32 bit	0.01	6.26	2.41GB	23.16	-	-
1MB ASCII Document Search	1M	10	8 bit	0.01	18.72	7.21GB	69.37	-	-
1% Human Genome Search††	30M	50	2 bit	0.02	21.65min	49.39GB	1.32h	-	-

†Per each 10 characters of the text as reported in [Fri09]. ‡ State-of-the-art GC+ORAM Impl. [WCS15]. ††In practice, processing properly chosen 1% of human genome yields comparable accuracy to processing the entire genome (3B letters) [GS06].

7.4 Related Work and Comparison

There are several studies addressing secure string search (or pattern matching) due to its significance. A number of earlier work consider securing against malicious and covert adversaries [HL08, HT10]. They incur a significantly larger overhead compared to PriSearch. Such a comparison is not always straightforward or fair due to the difference in the adversary models. Thus, we report the comparison of PriSearch to protocols that are secure against the honest-but-curious adversary model.

De Cristofaro et al. introduced a secure pattern matching protocol based on Additively

Homomorphic Encryption (AHE) for genomic applications [DCFT13]. They have implemented their protocol based on two different AHE libraries, namely AH-ElGamal and EC-ElGamal. The most expensive part of their approach is initial encryption of the genome data which as they have reported takes 115 and 2,580 hours using AH-ElGamal and EC-ElGamal respectively for genome data with 3 billion letters. An additional 2.7 hours and 8.7 hours of data transmission are required given a 1Gbps link. In contrast, our approach takes almost 80 hours for finding a pattern considering the same text size and communication bandwidth. Besides, they (and also [RDGK16b]) consider searching in a particular predetermined part of the genome, whereas, we search the entire genome.

Baron et al., propose to reduce the problem of substring search to a series of linear operations (e.g., inner products and matrix multiplication) such that the operations can be efficiently computed using AHE [BEDM$^+$12]. As shown in Table 7.3, PriSearch is 2$\times$ faster compared to their approach.

A number of studies [Ver11, HT10] rely on oblivious polynomial evaluations to perform binary substring search. However, they cannot search on non-binary alphabet like genomic data. Similarly, the method proposed in [YSK$^+$13] can only find patterns in binary vectors and is based on *somewhat homomorphic* encryption.

Another approach is to construct an automaton based on the query and then obliviously evaluate it on the text as proposed in [TPKC07, Fri09]. The method in [Fri09] is two to three orders of magnitude faster than the one proposed in [TPKC07] and as shown in Table 7.3, our method is 15$\times$ more efficient than [Fri09].

Please note that we cannot utilize the "Early Termination" technique [RSS$^+$17] since the runtime of the protocol would reveal the index of the match which as a result reveals the query.

7.5 Summary

We introduce PriSearch, an efficient approach for secure string search where both query and text from two parties are kept private. Our approach is based on the provably secure Garbled Circuit protocol. We report $2\times$ performance improvement compared to the state-of-the-art solution [BEDM$^+$12]. We design a new string search algorithm that makes the access pattern to the text strictly in order, enabling us to search large texts without using expensive Oblivious RAM. The experimental results demonstrate the efficiency of PriSearch and its applicability to real-world problems.

7.6 Acknowledgements

This chapter, in part, has been published at the Proceedings of 2017 Design Automation Conference (DAC) and appeared as: M. Sadegh Riazi, Ebrahim M. Songhori, Farinaz Koushanfar, "PriSearch: Efficient Search on Private Data". The dissertation author was the primary author of this material.

Chapter 8

CAMsure: Secure Content-Addressable Memory

We introduce CAMsure, the first realization of secure Content Addressable Memory (CAM) in the context of approximate search using near-neighbor algorithms. CAMsure provides a lightweight solution for practical secure (approximate) search with a minimal drop in the accuracy of the search results. CAM has traditionally been used as a hardware search engine that explores the entire memory in a single clock cycle. However, there has been little attention to the security of the data stored in CAM. Our approach stores distance-preserving hash embeddings within CAM to ensure data privacy. The hashing method provides data confidentiality while preserving similarity in the sense that a high resemblance in the data domain is translated to a small Hamming distance in the hash domain. Consequently, the objective of near-neighbor search is converted to approximate lookup table search which is compatible with the realizations of emerging content addressable memories. Our methodology delivers on average two orders of magnitude faster response time compared to RAM-based solutions that preserve the privacy of data owners.

8.1 Introduction

The ongoing development of technology has led to the generation of a massive volume of data. Today's cloud servers contain a database of information that belongs to users (clients) across the world. Such central storage of information enables clients to search for their content of interest in the database. This user-cloud search model appears in a broad variety of applications such as online recommender systems [EBVL11], face recognition [SSW09], secure biometric authentication [BBC+10], to name a few. For example, in online dating websites, the server maintains a database of all clients' profiles. The search algorithm aims to find the most similar profile in the database that best matches a specific query. The output of the search is the ID of the most similar profile. In this context, the search algorithm is approximate by nature since the similarity metric is defined heuristically.

The database might contain private and sensitive information, hence, the data has to be handled securely. For instance, in the case of online dating websites, users may prefer not to disclose their private attributes to the cloud server. As another example, consider biometric authentication where the fingerprint of a user is captured and matched against the valid profiles on the server. If an attacker can hack the server and get access to the database, the security of the system and the privacy of users are both diminished. It is worth mentioning that, once the sensitive data is compromised, the attacker can use this information to fool any other fingerprint-based authentication system. Recent data breaches of giant Internet companies such as Yahoo [yah17] and Google [goo17] have demonstrated that cloud servers are vulnerable to internal and external attacks. Therefore, scalable methodologies should be developed, both in software and hardware, to guarantee the security of the query, search result(s), and the data stored on the server.

The existing solutions for secure approximate search mostly involve computationally expensive cryptographic operations. The underlying cryptographic protocols of these schemes are either Homomorphic Encryption [QA08] or Yao's Garbled Circuits [SHSK15]. The high

computation and communication burden of these methods hinder their practical implementation for data-intensive applications. Order-Preserving Encryption (OPE) [BCLO09] allows comparison over encrypted. However, NNS solutions based on OPE and Deterministic Encryption (DTE) are proven to be inseucre by Naveed et al. [NKW15]. Protocols based on Asymmetric Scalar-Product-preserving Encryption (ASPE) [WCKM09] have also been proven to be insecure against the Chosen Plaintext Attack introduced by Yao et al. [YLX13]. Another line of research focuses on encryption-free lightweight randomized embeddings that provide security at the cost of low search quality [BR11]. This solution adds specific noise to the raw data in order to reduce the information leakage. Although this approach is computationally less intensive, the added randomness significantly reduces the search accuracy.

This chapter proposes CAMsure, a lightweight solution for securing emerging CAMs in the context of approximate search. Instead of writing the raw data on CAM, our approach stores distance-preserving hash embeddings to provide data privacy. The hashing scheme that is utilized in CAMsure is called Locality Sensitive Hashing (LSH) [IM98a]. This family of hashing methods creates a randomized embedding of data while preserving the pairwise similarity. LSH was initially proposed to reduce the computational complexity of search by converting high-dimensional data into a compact binary string (hash value). However, it is shown that original LSH schemes are vulnerable to certain attacks as a malicious party can infer some information about the secret data based on its hash value [RCS+16]. The vulnerability originates from the fact that LSH schemes preserve the pairwise similarity of *any* two input values. Recently, a new secure LSH has been proposed to mitigate the information leakage [RCS+16]. The hash values generated from this new LSH have a low Hamming distance only if the original elements are very similar (more than a specific threshold T_S). As a result, they are secure against known attacks.

LSH translates the problem of approximate search to finding a hash within the database that has the smallest *Hamming Distance* (HD) to the query. The theory of LSH guarantees that similar elements in the data domain will have a small Hamming distance in their corresponding

hash embeddings (with a high probability). The approximate lookup nature of emerging CAM technologies can enable approximate search on the hash values. As a result, CAMsure can be realized with a minimal alterations in CAM [RGC+15], i.e., voltage overscaling. Our explicit contributions are as follows:

- Introducing CAMsure, the first realization of secure in-memory computation using CAM in the context of approximate search. Our proposed secure CAM supports dynamic insertion and deletion of records without compromising system efficiency and privacy of data owners. CAMsure ensures data confidentiality even if the database is compromised.
- Proposing an end-to-end system that connects state-of-the-art hashing schemes to content-based memory architectures. Our methodology avoids costly computations of previously proposed secure search solutions by leveraging in-memory computation techniques. Another feature of this approach is that it allows lightweight deployment of secure search over distributed networks.
- Providing comprehensive analysis on search latency, power consumption, precision, and privacy guarantees of CAMsure and illustrating the practicality of our approach on a real-world dataset. As opposed to conventional RAM-based solutions, CAMsure has single cycle search latency, resulting in up to two orders of magnitude faster search while preserving the privacy of data owners.

8.2 Methodology

8.2.1 Scenario and Privacy Concerns

We assume that the CAM architecture is employed in a cloud server to store the database. This database holds secret data from certain parties that we call *data owners*. We refer to each record of the database as a *word*. A user wants to search the database for words that are most

similar to her query. Traditionally, the client's query and the words are revealed to any party that has access to the database. Storing the raw sensitive data on CAM memories in a centralized fashion makes users susceptible to internal attacks (i.e. server acting as a malicious party) or external threats (i.e. server compromised by a hacker).

In our computational model, the sensitive data is stored in the secure CAM (database) while the meta-data and user's ID are stored in plaintext by the cloud server in a regular memory. For example, in an online dating application, the sensitive data includes user's age, ZIP code, physical attributes, and personal preferences of the person that he/she prefers to be matched to. The metadata includes the username, hash of the user's password, and user's ID in the database. The metadata is used to connect two persons whose profiles are matched. Note that this kind of metadata is the least possible information that has to be kept by the cloud server in order to operate such centralized web-based applications. The focus of this work is to create the first of its kind secure CAM.

Threat model: This chapter assumes that the owner of the CAM (the server) is *untrusted*. More precisely, CAMsure is secure against honest-but-curious (semi-honest) server. In this attack model, server may attempt to infer information about the data that she stores, sends, and receives but it is assumed that the server will follow the protocol. This threat model strongly satisfies today's privacy concerns where users wish to hide the content of their query from the cloud server. Since we do not trust the server, the privacy of CAM words, the query, and also the search result should be preserved. This threat model also covers the situation where the entire database is compromised by an attacker.

To provide security for the CAM architecture, the users employ a hashing technique that hides the content of their data. The hashing method preserves similarity, meaning that data points that are close in the data domain are also similar in the hash domain, and those that are not close in the data domain are dissimilar in the hash domain. Section 8.2.2 describes, in high-level, the methodology of CAMsure for preserving the confidentiality of CAM data.

8.2.2 CAMsure Overview

CAMsure delivers scalable and lightweight privacy-preserving approximate search. As depicted in Figure 8.1, the scenario of CAMsure comprises two main ingredients: (i) a hashing scheme called LSH and (ii) a database consisting of one (or several) CAM blocks.

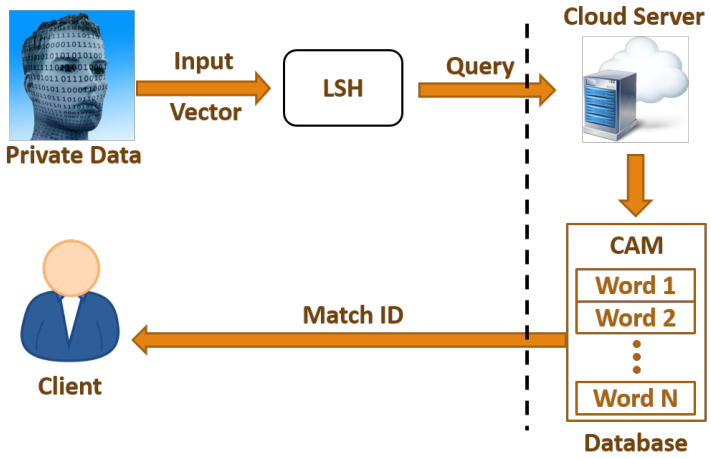


Figure 8.1: Overview of the secure approximate search scenario using CAMsure.

On the client side, the input is hashed using LSH to create a query which hides the actual content of the input vector. LSH takes as input a vector of real-valued elements and maps it into a binary string. The amount of information that the hash reveals about the actual input can be controlled as we discuss in section 8.3.5.

On the server side, the hash values provided by data owners are stored in the database. The database is realized by a certain number of CAMs. Due to the resemblance-preserving property of LSH, server can search for similar hash words within the database without actually knowing what the query or the words correspond to; therefore, CAMsure provides the security of CAM contents against malicious parties. In addition to providing security, CAMsure reduces the computational complexity of approximate search by converting high-dimensional vectors into hash values. It supports dynamic insertion and deletion of database words since the hash embeddings are generated independently. To the best of our knowledge, CAMsure provides the first secure CAM for the purpose of approximate search. We elaborate on the concept of LSH

in section 8.2.3. In Section 8.2.6, we discuss the additional modifications required to support approximate search in CAMs.

8.2.3 Hashing Methodology

In this section, we illustrate the hashing method that is utilized in CAMsure. Figure 8.2 depicts the concept of LSH for secure approximate search. The left-hand side belongs to the data domain where each input is represented as a feature vector of real values, e.g., features extracted from an image of the person's face. The right-hand side represents the hash domain where each word is a binary string computed from the original data. The two hash strings in Figure 8.2 are different since the original feature vectors (the faces) are dissimilar. Any LSH scheme guarantees that the probability of two LSH bits colliding (being equal) is a monotonic function of the similarity in the original domain. For example, assume that LSH guarantees a collision probability of 95% in the hash domain for a similarity threshold of $T_S = 0.9$ in the data domain. This implies that, on average, $0.95 \times l$ bits should match for l-bit LSH embeddings of two inputs with a similarity of $t = 0.9$. Therefore, utilizing LSH enables us to translate the near-neighbor search to finding the words in the database that have Hamming Distance of $(1 - 0.95) \times l$ or less. Although this approach introduces a certain degree of inaccuracy, the added error rate is negligible in practice [IM98b]. We elaborate on the accuracy of CAMsure in Section 8.3.4 comprehensively.

The key achievement of using LSH is that the server no longer needs to compute pairwise similarities using a processor. The near-neighbor search is performed by just checking if the incoming query has small Hamming distance (given certain threshold) to any of the database words. This check can be efficiently done in only *one* clock cycle by minimal modification in the CAM [RGC+15].

As we discussed, MinHash and SimHash require a set of randomly generated numbers, i.e. MinHash requires random permutation and SimHash requires random projection vector, which

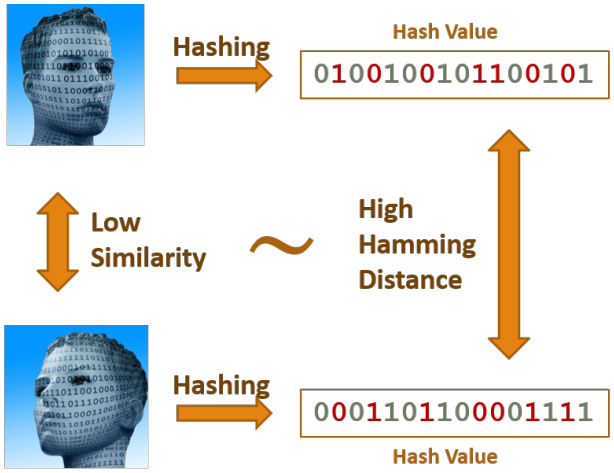


Figure 8.2: Relationship between the data and hash domains. The hashing scheme preserves the proximity of inputs only for similar data.

we refer to as *random seeds*. Although these seeds are generated randomly for each LSH bit, they have to be consistent for generating the hash of all the data in the database and also for the incoming query. To satisfy this requirement, the server generates these seeds and announces them publicly. Whenever a new user wants to search the database, she needs to download these seeds, compute the hash of her input, and send the hash to the server. The size of these seeds is very small, usually in the order of KB and is especially suited for Internet settings. This is in contrast to secure multiparty computation methods which require very high communication bandwidth (in the order of MB or GB) [SHSK15]. It is worth mentioning that these seeds have to be downloaded only once at the time when the user registers to the cloud server.

8.2.4 Data Leakage of Traditional LSH Schemes

The amount of information that one can infer from a single LSH embedding and the publicly available random seeds is limited and is discussed in Section 8.3.5. However, an attacker can generate multiple fake input vectors and perform the *triangulation* attack as introduced in [RCS+16, Ria16]. The susceptibility comes from the fact that traditional LSH schemes allow an estimation of the similarity between original vectors corresponding to any pair of hash values. Assume that an attacker wants to reconstruct the original vector v hashed into $hash(v)$. He can

create random vectors v_i, compute their hashes $hash(v_i)$, and compare them to $hash(v)$. Since traditional LSH methods preserve the pairwise similarity for *any* two input vectors, the attacker can identify vectors v_i to which v is most similar according to the Hamming distances of $hash(v)$ and $hash(v_i)$. Consequently, the attacker reduces the subspace in which v can reside and estimates the elements of v up to a certain precision. Performing the same procedure iteratively results in a very good estimation of v. Therefore, if the server is malicious or an attacker can get access to the database, the privacy of all users is compromised.

Recently, a novel hashing technique is proposed to mitigate the triangulation attack [RCS+16]. The idea behind their method is to make sure that the collision probability of LSH bits is high only when the two inputs are *very similar* and is as low as 50% for non-neighbors (same as collision probability of two completely random bits). As a result, if an attacker repeats the same triangulation attack, he cannot estimate the original attributes by brute-forcing the possible input space. In the following section, we illustrate how the hashing method of [RCS+16] works and how it achieves security against the triangulation attack.

8.2.5 LSH Transformation

Authors of [RCS+16, Ria16] have proposed an LSH *transformation* that eliminates unnecessary information leakage of traditional LSH methods. To perform this task, they suggest that the transformed hashing scheme should have the following property: for any two input vectors that have similarity lower than a threshold (T_S), their corresponding hashes must have very high Hamming distance. As a direct consequence of this condition, for any two dissimilar inputs, the collision probability of any single LSH bit must be close to 0.5, making them indistinguishable from two randomly generated bits. In other words, the correlation between two hashes must be very close to zero in order to avoid information leakage.

The proposed transformation which satisfies this condition is

$$h^{1-bit}_{secure}(x) = h_{univ}(h_1(x), h_2(x), ..., h_k(x)), \tag{8.1}$$

where $h_i,\ i \in \{1,\ 2, ...,\ k\}$ are k independent hash functions (with independent initial random seeds) and $h_{univ}(.)$ is defined as

$$h_{univ}(\alpha_1,\ \alpha_2, ...,\ \alpha_k) = (r_{k+1} + \sum_{i=1}^{k} r_i \alpha_i)\ mod\ p,\ mod\ 2, \tag{8.2}$$

where $r_i,\ i \in \{1,\ 2, ...,\ k\}$ are randomly generated integers and p is a prime number. Note that r_i should remain fixed for all of the hash computations once they have been derived. Parameter k is called the *security parameter*. h_i can be implemented as any traditional LSH function, e.g., MinHash or SimHash.

The effect of this transformation is that the collision probability of two 1-bit hashes, $h^{1-bit}_{secure}(x)$ and $h^{1-bit}_{secure}(y)$, drops rapidly as the similarity of x and y decreases. For instance, the collision probability of secure MinHash is

$$Probability\ \{h^{1-bit}_{secure}(x) = h^{1-bit}_{secure}(y)\}\ =\ \frac{\mathcal{R}^k + 1}{2} \tag{8.3}$$

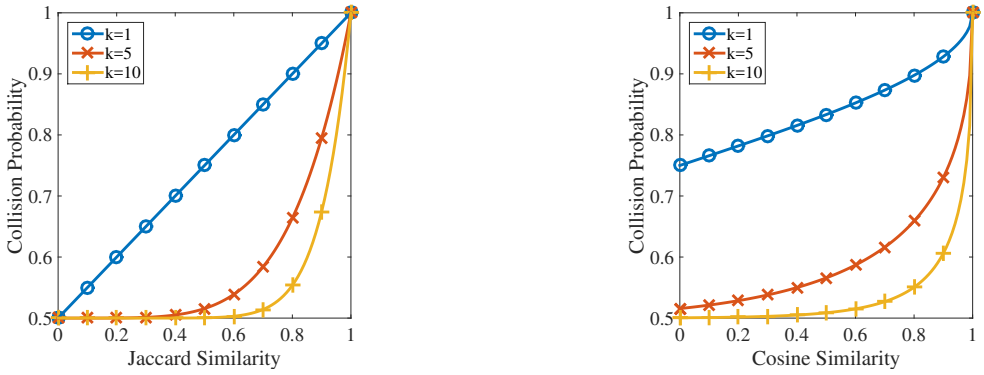
where $\mathcal{R}$ is the Jaccard similarity between x and y. Similarly, the collision probability of secure SimHash is

$$Probability\ \{h^{1-bit}_{secure}(x) = h^{1-bit}_{secure}(y)\}\ =\ \frac{(1 - \frac{\cos^{-1}(\mathcal{C})}{\pi})^k + 1}{2} \tag{8.4}$$

where $\mathcal{C}$ denotes the Cosine similarity of x and y. The formal proofs of Equations 8.3 and 8.4 are provided in [RCS+16].

We have depicted the collision probability of secure MinHash and SimHash for different security parameters, $k \in \{1, 5, 10\}$, in Figure 8.3. Note that $k = 1$ is identical to traditional LSH functions (please see Equation 8.1) and represents the baseline for comparison with traditional

LSH schemes.



(a) MinHash (b) SimHash

Figure 8.3: Collision probability vs. similarity for MinHash and SimHash functions for three different security parameters $k \in \{1,5,10\}$. $k = 1$ represents traditional LSH methods.

As mentioned in Section 8.2.4, the triangulation attack is possible for traditional LSH schemes since they allow a similarity estimation for *any* two input vectors whose similarity ranges from 0 to 1. This is shown as the curves corresponding to $k = 1$ in Figure 8.3. As can be seen, an attacker can estimate how similar vectors v_i are to the secret vector v by comparing the HD between $hash(v_i)$ and $hash(v)$. v is most similar to the vector v_j whose Hamming distance to $hash(v)$ is minimum. However, the same property does not hold for $k = 5$ or $k = 10$ since the collision probability drops drastically as the similarity goes to zero. For example, assume v_j has a similarity of 0.8 to v, therefore, the collision probabilities for 1-bit secure SimHashes are 0.9, 0.66, and 0.55 for $k = 1,5,10$, respectively (see Figure 8.3). Please note that the collision probability of any two random bits chosen from the Uniform distribution is 0.5. The collision probability of hash bits directly translates to the expected number of bit-matches of the hashes.

Assuming l-bit hash embeddings, the number of bit matches between $hash(v_j)$ and $hash(v)$ are on average $0.9l$, $0.66l$, and $0.55l$ for $k = 1,5,10$, respectively. Since $0.9l$ is an outlier, the attacker gets to know that v_j must be very similar to v and he can iteratively continue his search in a smaller subspace. However, when the secure hashing function with parameter $k = 10$ is used, the attacker observers $0.55l$ bit matches (with a high probability). Since this number is

very close to the number of bit matches between any two random l-bit strings, the attacker cannot infer any information. In CAMsure, the privacy of data owners and data users is preserved since the server does not have access to the plaintext of users' data and the query. This means that even if the entire database is compromised or the server is malicious, users' data is not revealed. The comprehensive security analysis is provided in Section 8.3.5.

As we have illustrated in this section, in order to find near-neighbors of the query, one needs to find database words whose Hamming distance to the query is lower than a certain threshold. For example, if the threshold is 2, any word in the database with Hamming distance of 0, 1, or 2 to the query should be reported as a near-neighbor. In Section 8.2.6, we explain how one can perform the approximate search with HD tolerance in CAMs.

8.2.6 Approximate Search in CAMs

The probabilistic nature of LSH implies that even the most similar raw records might have some Hamming distance in their hash embeddings. Consequently, the lookup table search is no longer a search with exact matching. Any stored word whose Hamming distance to the input query is below some pre-defined threshold is considered a match (near-neighbor) and its corresponding match-line should trigger a high voltage. Therefore, the proposed CAM should be capable of performing search with certain HD tolerance.

The earliest implementation of CAM can only perform exact matching. Ternary CAMs (TCAMs) provide the additional functionality of having "don't care" bits in the seach key [MF93]. These bits are masked during the search, i.e., the masked CAM cells output a high voltage regardless of the corresponding bits in the input query. The "don't care" bits in TCAMs are not permuted, i.e. their location in the input string is fixed; thus, TCAMs cannot be readily used for evaluating the Hamming distance. Therefore, throughout this chapter, we do not focus on TCAMs.

Authors of [RGC+15] present a modified CAM that allows approximate search based

on Hamming distance. Figure 8.4 illustrates the idea behind their methodology for approximate matching. The solid red line corresponds to the match-line's transient voltage when the query matches the stored word with zero Hamming distance. The dashed lines correspond to match-lines with different Hamming distances. The HD tolerance can be tuned using two techniques:

- **Changing the sampling time:** It can be observed that, as the number of mismatched cells is increased, the output voltage drops more abruptly. This change in the decay time allows us to infer a partially mismatched data as a match by simply changing the sampling time; the earlier the output is sampled, the more Hamming distance is tolerated.

- **Voltage over-scaling:** Another method to tune the Hamming distance tolerance is to fix the sampling time and change the supply voltage applied over the cells. Note that decreasing the supply voltage also reduces the power consumption of the CAM.

Authors of [RGC+15] mention two downsides for the proposed approximate CAM: (i) possibility of false match and (ii) having multiple matches. Possibility of mismatch simply refers to having approximate matches, which in fact, is a design goal for CAMsure. Having multiple matches is also desired as the goal of near-neighbor search is not to identify a single record but to find all data points that are similar to the query. In addition to the aforementioned issues, their design is not capable of distinguishing Hamming distances that are higher than 2. This issue might be partially mitigated by increasing the capacitance of the CAM cells which can make higher Hamming distances distinguishable.

8.3 Analyses

In this section, we first demonstrate the superiority of CAM over RAM for the purpose of near-neighbor search. Next, we analyze the accuracy and security aspects of CAMsure.

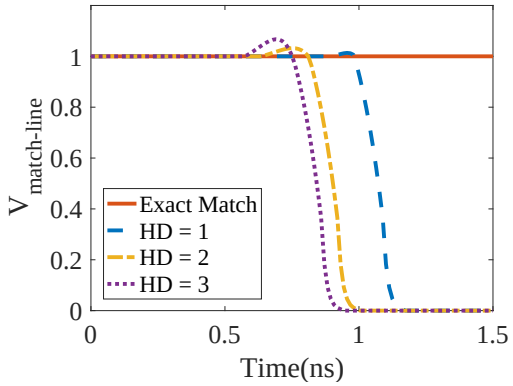


Figure 8.4: CAM match-line transient output voltage for exact and approximate matching.

8.3.1 Delay and Energy Analysis

We compare the proposed search method over hash embeddings stored in Static RAM (SRAM) and CAM architectures. Specifically, we compare the two architectures for different hash embedding widths (i.e. 32-bit and 64-bit) denoted by l and different sizes of the lookup table which we denote by N. For the SRAM architecture, we simulate a cache using the CACTI 5.3 tool [TMAJ] which is an open-source software from HP Co. to estimate the read-energy and search delay of different cache designs. Table 8.1 outlines the cache used in our reports. For the CAM-based implementation, we incorporate the energy and delay reports of [RGC$^+$15] to analyze the proposed methodology for memristive CAMs that are capable of approximate search. Specifically, we assume that multiple CAM blocks from [RGC$^+$15] with the corresponding bit-width are instantiated to accommodate the database; the search delay would be the same as reported in [RGC$^+$15], while the energy consumption increases linearly with respect to the number of instantiated blocks. We also compare commercialized CMOS-based CAMs from the LANCAM family [com17b, com17a]. Table 8.2 outlines the specifications of the CAM blocks.

Table 8.1: Cache design for the RAM baseline.

Cache Size	Line Size (bytes)	Associativity	No. Banks	Technology
Variable (based on lookup table size)	32	4	1	45 nm

Note that both SRAM and CAM are used to implement the same search algorithm, therefore, their search accuracy is the same; the only difference between SRAM and CAM is

Table 8.2: Specifications of CAM baselines.

CAM Architecture	Delay (ns)	No. Words	Technology (nm)	Search Energy (pJ)		
				Exact	HD = 1	HD = 2
32-bit resistive [RGC+15]	1.5	64	45	3.901	1.738	1.332
64-bit resistive [RGC+15]				4.568	1.953	1.479
32-bit CMOS [com17b]	70	4096	NA	1617	-	-
64-bit CMOS [com17a]	50	4096	NA	2475	-	-

in the way they perform the search. The following sections compare the delay and energy of a single search for SRAM and CAM.

8.3.2 Delay Analysis

We compare the search delay of different architectures in Figure 8.5. For the cache-based implementation, we change the cache size of the CACTI memory to accommodate the number of words within the table. We give an advantage to the baseline SRAM by assuming that the processing unit is fully pipelined and the memory can operate in maximum throughput. For this purpose, we use the "interleave cycle time" metric reported by the CACTI tool which we denote by T. We also assume that the cache hit rate is 100%. The number of sequential read operations to perform a single search is $\frac{N}{m}$ where N is the number of words within the database and m is the number of words fetched by a single read operation. More specifically, m is the number of words within one line of the cache

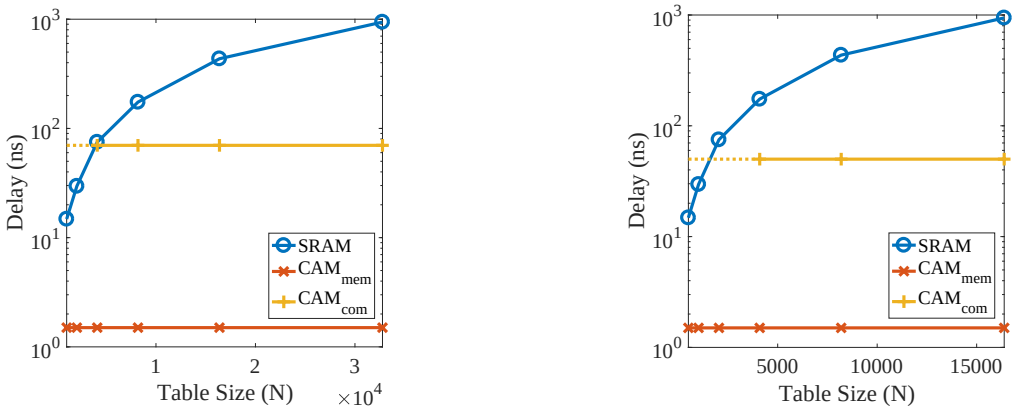
$$m = \frac{line\ size}{bytes\ per\ embedding}.$$

A 64-bit (32-bit) embedding requires 8 bytes (4 bytes) to be stored and assuming a line size of 32 bytes in Table 8.1, each interleaved access provides $m = \frac{32}{8} = 4$ ($m = \frac{32}{4} = 8$) words. The search delay reported in Figure 8.5 is then computed as

$$Delay = T\frac{N}{m}.$$

For the CAM-based table, we assume that multiple banks of the memristive and the CMOS-based blocks (Table 8.2) are instantiated to perform the search in parallel, thus, the CAM

search time is fixed for different database sizes. In contrast, the SRAM search time is increased for a higher number of words in the lookup table since the words are processed sequentially. The results show that, compared to SRAMs, CAM architectures reduce the search delay by *two orders of magnitude*.

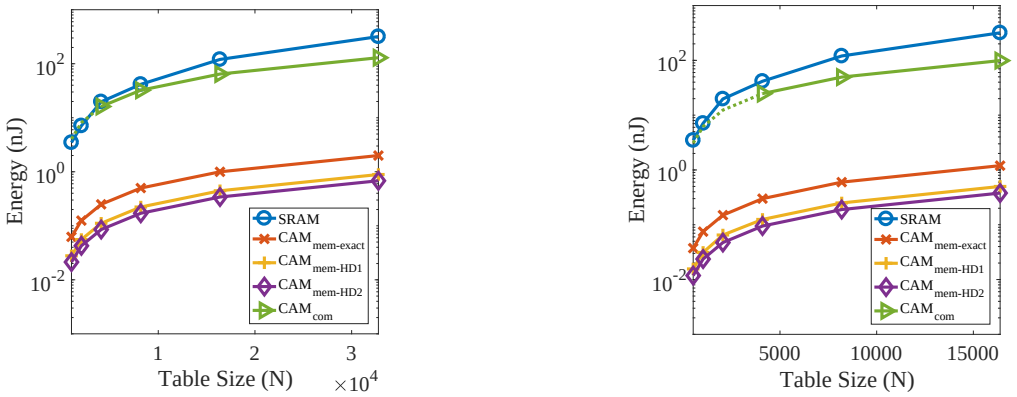


(a) 32-bit hash embeddings (b) 64-bit hash embeddings

Figure 8.5: Delay analysis for hash embeddings of size 32 and 64-bit. CAM_{mem} and CAM_{com} denote memristive and commercial CMOS-based CAMs, respectively. The dotted line for CAM_{com} shows interpolated data.

8.3.3 Energy Analysis

Figure 8.6 compares the energy consumption of different memory architectures. We take the "dynamic energy per read port" metric provided by the CACTI tool and multiply it by the number of sequential read operations to compute the search energy for SRAM. For CAMs, we assume that the search energy is linearly increased with respect to the number of CAM blocks. It is observed that the search energy of CAM is significantly lower than that of SRAM. Memristive CAMs offer a smaller energy consumption than CMOS-based CAMs. It is noteworthy that the power consumption is decreased as we increase the Hamming distance. A high HD tolerance can be achieved by lowering the supply voltage which in-turn reduces the search energy.



(a) 32-bit hash embeddings

(b) 64-bit hash embeddings

Figure 8.6: Energy consumption analysis for hash embeddings of size 32 and 64-bit and different Hamming distances. CAM_{mem} and CAM_{com} denote memristive and commercial CMOS-based CAMs, respectively. The dotted line for CAM_{com} shows interpolated data.

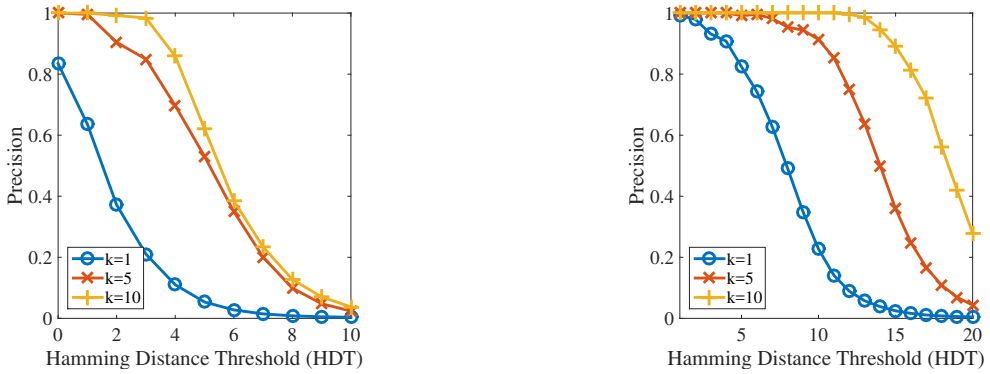
8.3.4 Accuracy Analysis

The performance and power analyses in Section 8.3.1 are general and account for any dataset regardless of the application. However, the search accuracy analysis which we discuss in this section is inherently dependent on the underlying dataset. For this purpose, we focus on the Speed-Dating [FIKS06] dataset which contains 8378 text survey samples (profiles). Each profile is represented as a 190-dimensional vector. Each element of the vector is a feature that contains sensitive information about the individuals, e.g., gender, race, age, the field of study, zipcode, income, etc. We report the search accuracy by comparing the result from CAMsure with the most similar elements in the original domain (before hashing) that pass similarity threshold of 0.95. Therefore, both the ideal search result and the result from CAMsure can be viewed as a bag of indices (of the database) which are claimed to be similar. CAMsure outputs the index of any word in the database that passes the Hamming Distance Threshold (HDT). In other words, any word that has Hamming distance of HDT or less to the query is reported as a near-neighbor.

We report the accuracy results based on two metrics: *precision* and *recall*. *Precision* is defined as $\frac{n_c}{n_r}$, where n_r denotes the number of total near-neighbor indices reported by CAMsure and n_c denotes the number of indices reported by CAMsure that are correct (truly similar). Here, we consider two points with similarity measure of 0.95 or higher to be called near-neighbors.

Recall is defined as $\frac{n_c}{n_t}$ where n_t is the total number of words within the database that are considered as near-neighbors of the query. An ideal system should report *all and only all* of the true near-neighbors which means both precision and recall should ideally be equal to one. A trivial solution that outputs all entries in the database has high recall (=1) but unacceptable precision ($\sim$0). We generate 32-bit and 64-bit LSH embeddings for each profile in the dataset. We randomly select the query as one of the profiles from the dataset and repeated the experiment for 100 different queries for each set of parameters.

Figure 8.7 illustrates the precision results as a function of the HDT for different security levels (k), where two plots are depicted for 32-bit and 64-bit LSH embeddings, respectively. The number of bits in the LSH embedding is equal to the bit-width of each word stored in CAM. The search precision is higher for low values of HDT. This is persistent with our intuition since the HDT defines the measure of search accuracy: by choosing higher values for the HDT, more words are selected from the database that may not correspond to a true near-neighbor, reducing the overall precision.



(a) 32-bit hash embeddings (b) 64-bit hash embeddings

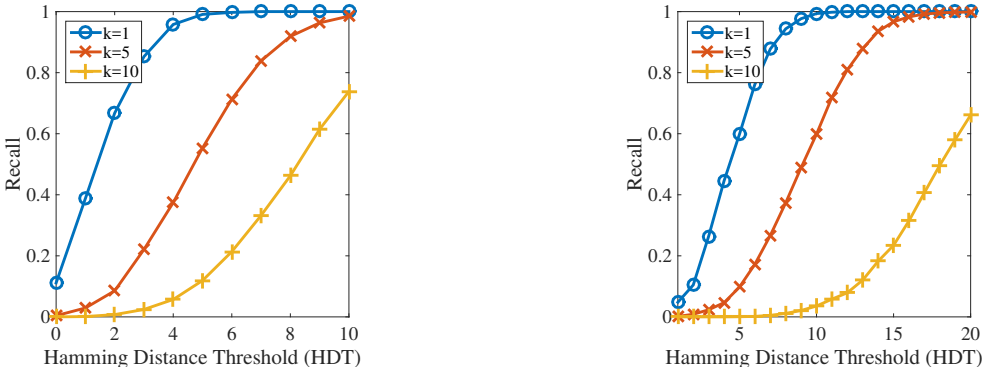
Figure 8.7: Precision analysis for hash embeddings of size 32 and 64-bit for different security parameters (k).

Figure 8.8 provides the recall results as a function of the HDT for different security levels (k) and different LSH embeddings (32-bit and 64-bit). In contrast to the precision reports, increasing the HDT results in a higher recall. Higher values of HDT allow for more database

words to be selected. Therefore, there is a trade-off between precision and recall which is a known concept in near-neighbor search algorithms.

It is beneficial to plot the precision as a function of the recall in order to illustrate the trade-off (Figure 8.9). As the analysis suggests, the hashing scheme utilized in CAMsure achieves a reasonably high precision and recall while providing data confidentiality.

Two observations are worth mentioning: first, 64-bit LSH embeddings deliver better precision/recall trade-off with the cost of higher computational complexity (see Figure 8.9). Second, increasing the security parameter k comes with the cost of lower precision and recall. Therefore, the number of bits in the hash embedding (l) and the security parameter (k) are the two factors that should be tuned to utilize CAMsure for different datasets.

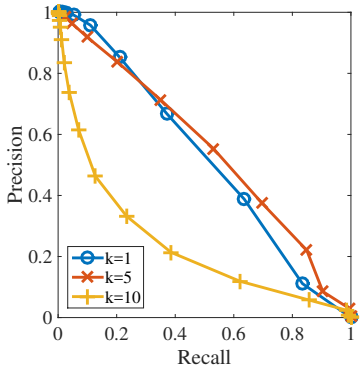


(a) 32-bit hash embeddings (b) 64-bit hash embeddings

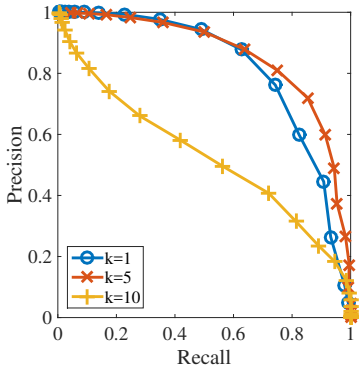
Figure 8.8: Recall analysis for hash embeddings of size 32 and 64-bit for different security parameters (k).

8.3.5 Security Analysis

We provide a comprehensive security analysis of CAMsure in the event where the cloud server is malicious or the database is compromised by an attacker. We analyze, both theoretically and experimentally, the scenario where an attacker wants to infer as much information as possible about one or multiple words in the database. Assuming that the attacker has complete access to the database, we are interested in knowing what information can be inferred from the original



(a) 32-bit hash embeddings



(b) 64-bit hash embeddings

Figure 8.9: Precision vs. Recall analysis of 32 and 64-bit hash embeddings for different security parameters (k).

data given the hash embeddings. In Section 8.3.6, we explain the state-of-the-art method, called *compressed sensing*, that aims to reconstruct the original data given its hash. In the second analysis, we consider the idea of the *brute-force* attack on CAMsure. We provide theoretical and experimental analysis for this attack. The third analysis focuses on validating the theory of LSH transformation. Finally, we focus on a more sophisticated attack, called *Triangulation* [RCS+16], which aims to extract information by leveraging the correlation between hash embeddings of multiple data points scattered in different spans of the space.

8.3.6 Compressed Sensing

The theory of compressed sensing aims to extract information from randomized embeddings. However, there is currently no practical approach that can extract meaningful information from the LSH embeddings utilized in this chapter. In general, compressed sensing requires more than $\theta(s\log_2 D)$ measurements to provide reasonable accuracy [CW08] where s and D are the number of non-zero elements and the dimensionality of the input vector, respectively. However, in CAMsure, the bit-width of the hash embedding (32 or 64) is far smaller than the aforementioned lower bound which is at least $190 \times log_2(190) = 1438$ bits for the Speed-Dating dataset. Compressed sensing algorithms are similar to the idea of triangulation attack which we discuss in Section 8.3.9.

8.3.7 Brute-force Attack

The attacker might attempt to guess the original value of a vector and validate his choice by comparing the hash value of his guess with the database entries. If he finds a hash value identical to what he has computed, the original data vector is revealed to him. However, this attack is computationally impossible as we describe next. If we represent each real number as a fixed point 32-bit, a D dimensional vector input has

$$N_{possible} = (2^{32})^D$$

possible different values. According to NIST standard [nis17], any brute-force attack that requires 2^{128} or higher operations is considered infeasible. Since $N_{possible}$ grows *exponentially* with D, even for small dimensionality ($D = 4$ or higher), the attack is infeasible.

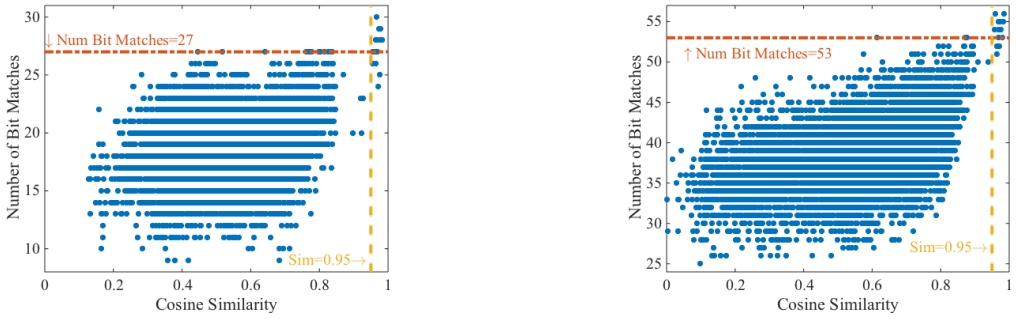
8.3.8 Validating the LSH Transformation

The theory behind the LSH transformation of [RCS+16] suggests that mutual information between the bits of two hashes drops rapidly as the similarity of the original vectors is decreased. For example, in the case of SimHash,

$$I(h^{1-bit}_{secure}(x); h^{1-bit}_{secure}(y)|\theta) < (1-\frac{\theta}{\pi})^k log(\frac{1+(1-\frac{\theta}{\pi})^k}{1-(1-\frac{\theta}{\pi})^k}), \tag{8.5}$$

where $I(.)$ denotes the mutual information [CT12] between two SimHash bits and $\theta = \cos^{-1}(\mathcal{C})$ with $\mathcal{C}$ being the Cosine similarity, defined in Equation 2.5. In this section, we validate this theory experimentally. We choose a random profile from the Speed-Dating dataset as our query and consider its similarity to all other profiles in the database. In particular, we compute the true similarity measure in the data domain $s_{original}$ and the number of matched bits in the hash domain s_{hashed} using a security parameter of $k = 5$ for 32-bit and 64-bit embeddings. Figure 8.10 depicts

s_{hashed} as a function of $s_{original}$ between the chosen profile and all other profiles in the database.



(a) 32-bit hash embeddings (b) 64-bit hash embeddings

Figure 8.10: Validating LSH transformation in practice on Speed-Dating dataset for security parameter $k = 5$.

The vertical yellow lines in Figure 8.10 show the similarity of 0.95. The horizontal red lines depict 27 and 53 number of bit matches for 32 and 64-bit LSH embeddings, respectively. As our experiments show, the results closely follow Equation 8.5. If two profiles are not similar in the original vector representation (x-axis), the corresponding tuple ($s_{original}$, s_{hashed}) resides on the left-hand side of the yellow line. It can be seen that the corresponding hashes have a random number of bit matches anywhere between zero to $HDT = 27$ ($HDT = 53$) for such profiles. Meanwhile, two profiles with a similarity of 0.95 or higher reside on the right-hand side of the yellow line and the corresponding number of bit matches is higher than the threshold $HDT = 27$ ($HDT = 53$). This is consistent with our expectation from the characteristic of the LSH transformation.

8.3.9 Triangulation Attack

A more sophisticated attack is to create multiple random vectors, compute their hashes, and triangulate the secret vector by comparing the HD of the corresponding hashes (see Section 8.2.4). However, the attack is not effective for the LSH transformation that we have utilized as we explain here.

Each D-dimensional vector can be viewed as a point in the D-dimensional space. As

the dimensionality increases, the volume of space grows exponentially. This phenomenon is known as *curse of dimensionality* [KM11]. The distance of two randomly chosen points in a high-dimensional space is much higher than that of a low-dimensional space. To Illustrate this phenomenon in practice, we randomly choose a point p in the 190-dimensional space (the same dimensionality as the Speed-Dating dataset). After that, 10^7 random points have been created and the mutual Cosine similarities between each one of these and p have been computed. Figure 8.11 shows the statistical distribution of mutual similarities. We have repeated the experiment for 10 different random choices of p. Based on experimental results on 10^7 different points, the values for similarities range from -0.37 to 0.38 and not even one point has a similarity higher than 0.4. Please note that the attacker needs to obtain the mutual similarity of 0.9 or higher (see Section 8.2.5) to successfully estimate the true vector corresponding to a given hash. However, as is illustrated in Figure 8.11, an attacker simply cannot find the starting vectors and as a result, the attack is not effective.

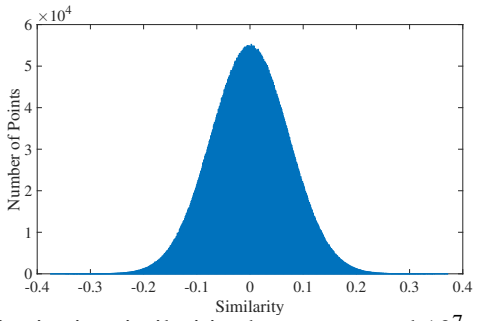


Figure 8.11: Histogram of pairwise similarities between p and 10^7 randomly selected points in 190-dimensional space.

8.4 Related Work

To the best of our knowledge, there has been no attempt to secure the search operation in content addressable memories. Here, we briefly describe previous solutions to secure near-neighbor search problem that use the convectional Von Neumann processing architecture. All

of these methods require a processing unit at the server side and incur high computation and communication while CAMsure can process the query in *real time*. We categorize previous works into three main groups:

Differential Privacy (DP): This line of research provides certain privacy guarantees by adding specific kind of noise to the data that is stored on the server. The noise is added such that the statistical properties (e.g. average, variance, etc.) of the whole database are preserved while individual words are altered [Dwo08]. This privacy enhancing approach is useful in scenarios where the clients are interested in statistical properties of the database not the individual records within the database. In addition, the security model of DP is different; the assumption is that the server which holds all users' data is *trusted* and an attacker only has the ability to query the database and infer as much information as possible based on the results that he receives. In contrast, our assumption is that the server is not trusted. Therefore, CAMsure provides stronger security and privacy guarantees.

Property-Preserving Encryptions: As we discussed in Section 8.1, secure NNS solutions based on Order-Preserving Encryption (OPE) [BCLO09], Deterministic Encryption (DTE), and Asymmetric Scalar-Product-preserving Encryption (ASPE) [WCKM09] are proven to be insecure by [NKW15] and [YLX13], respectively. While Order Revealing Encryption (ORE), Inner Product Encryption (IPE), and Hidden Vector Encryption (HVE) are appropriate candidates for securing conventional RAM-based databases, they are not compatible with the CAM architecture. Another line of research is based on the searchable encryption [PKV$^+$14]. However, this type of solutions can only answer exact query matches and not the near neighbors.

Noise-Addition Techniques: Another idea is to add noise to the query *before* sending it to the server in order to protect the privacy of data owners [MMSK18, MML$^+$19, GMGM13]. Authors in [BR11] have looked into the possibility of such idea. They observe that adding noise introduces a privacy/utility trade-off. In other words, if the user adds stronger noise, an attacker will infer less information about the original data but as a consequence, the precision of search

is sacrificed. This approach has very limited practical usage as a reasonable privacy guarantee results in a very poor precision for the near-neighbor search and sabotages one of the main goals which is the search quality. In contrast, we have shown in Section 8.3.4 that our approach delivers high precision/recall for secure approximate search.

Secure Function Evaluation (SFE) protocols: SFE protocols allow two or multiple parties to evaluate any function on their private inputs. There are two possible ways to deploy SFE protocols for secure search: (i) the server possess the entire database in plaintext and acts as the first party and the user who wants to query the database acts as the second party [RSK17b, RDGK16a, HRK18]. In this scenario, the server and the user engage in a secure *two-party* computation where the database and the user's query remain private. However, it is assumed that the server already knows all the data in plaintext which is in contrast to our security model. (ii) All of the data is not centralized and it is kept by the data owners only. To process each query, all parties need to engage in a secure *multiparty* computation. In this case, the privacy of all data owners is guaranteed but it is necessary for all data owners to be online and have a peer-to-peer pre-established communication channel which is not a realistic assumption.

SFE protocols preserve the privacy of engaging parties completely and they do not leak any information about the inputs. Unfortunately, all of these protocols require massive computation time, communication bandwidth, and several rounds of communication between the engaging parties. The aforementioned drawbacks make SFE protocols to have very limited practical usage.

Recently, authors of [SHSK15] have proposed to deploy Yao's Garbled Circuits protocol (one of the most efficient SFE protocols) for the privacy-preserving k-nearest neighbors problem. They consider the two-party computation model (scenario i). They report 2.54 GB communication between two parties and the execution time of 6.7s for 1 Gbps communication bandwidth when the database contains $128,000$ records. CAMsure only requires the user to locally compute the hash of her data (negligible time) and send this data to the cloud server (transmitting less than a KB of data) while the query can be answered in one clock cycle on the server.

8.5 Summary

This chapter proposed CAMsure, the first realization of secure content addressable memory in the context of approximate near-neighbor search. CAMsure utilizes state-of-the-art hashing methods to translate the near-neighbor search algorithm to approximate table lookup. Our comprehensive security analysis shows that CAMsure preserves the confidentiality of data even when the memory is compromised by a malicious party. The proposed methodology is compatible with emerging CAM technologies. The security of content addressable memory is critically important since CAMs are tightly coupled with the main processing units for in-memory computing, aiming to enhance the efficiency of modern computers. Our analysis shows that CAMsure can improve the runtime of RAM-based implementations by up to two orders of magnitude while providing data security. Expansion of the proposed method for applications other than near-neighbor search is another direction to follow in future.

8.6 Acknowledgements

This chapter, in part, has been published at the 2017 ACM Transactions on Embedded Computing Systems (TECS) and appeared as: M. Sadegh Riazi, Mohammad Samragh, Farinaz Koushanfar, "CAMsure: Secure ContentAddressable Memory for Approximate Search" [RSK17a]. The dissertation author was the primary author of this material.

Chapter 9

SynFi: Synthetic Human Fingerprints

Authentication and identification methods based on human fingerprints are ubiquitous in several systems ranging from government organizations to consumer products. The performance and reliability of such systems directly rely on the volume of data on which they have been verified. Unfortunately, a large volume of fingerprint databases is not publicly available due to many privacy and security concerns.

In this chapter, we introduce a new approach to automatically generate high-fidelity synthetic fingerprints at scale. Our approach relies on (i) Generative Adversarial Networks to estimate the probability distribution of human fingerprints and (ii) Super-Resolution methods to synthesize fine-grained textures. We rigorously test our system and show that our methodology is the *first* to generate fingerprints that are computationally indistinguishable from real ones, a task that prior art could not accomplish.

9.1 Introduction

Evaluating the performance and reliability of identification and verification fingerprint-based systems requires access to a large fingerprint database. However, in practice, obtaining a massive corpus of fingerprint images incurs a high cost. In many cases, the research groups that

are developing fingerprint-based authentication systems, do not have access to a large publicly-available database. The performance of these systems is directly dependent on the quality and quantity of the available data.

In addition to the above obstacles, gathering fingerprint impressions of a large population of people raises severe privacy and security concerns. In case of a breach, the fingerprint of many users will be directly exposed to attackers and can be used to fool any other authentication systems that accept fingerprints. To this end, we study the task of generating *synthetic fingerprints* which can solve the challenges mentioned above. Synthetic fingerprints solve the availability concern as they can be generated for virtually any number of samples. Moreover, synthetic fingerprints are artificially generated; hence, they do not leak any information about real identities.

Synthetic fingerprints also play essential roles in other tasks as well. For example, they can be used to analyze the robustness of a verification system against Trojan attacks [MMJP09]. To perform this analysis, a large number of fingerprints are needed where their fine-grained features can be varied while fixing other characteristics such as image orientation.

Fingerprints can be categorized based on their global structure and curvatures. Some of these categories are drastically rarer, and their synthetic counterparts can be used to compensate for the imbalance. One can generate a specific type of fingerprints that are rarer in the real-world. Moreover, the security of several biometric storage mechanisms that protect fingerprints in case of a breach relies on the assumption that the size of a database is bigger than a specific threshold [CRC$^+$19]. In these scenarios, synthetic fingerprints can be used as a means to populate small databases.

Prior synthetic fingerprint generation solutions were able to either create synthetic *templates* of fingerprint's micro-features or synthetic image of actual fingerprints but at a low resolution. Solutions based on mathematical models of fingerprints suffer from lack of entropy and generalization to accurate probability distribution of real fingerprints [CMM04]. Prior solutions based on Deep Learning (DL) models also cannot produce high-quality images due to the

small volume of the available real fingerprints to train these models [BRT+18].

In this chapter, we present SynFi, a new comprehensive framework to automatically generate high-quality synthetic fingerprints at scale. Our solution formulates the process of generating synthetic fingerprints as two parallel deep learning tasks based on Generative Adversarial Network (GAN) and Super-Resolution (SR) paradigm. In particular, SynFi formalizes and satisfies the following design goals to meet real-world expectations: (i) the generated samples should preserve the minutiae characteristics of fingerprints used for authentication systems, e.g., ridge structure, bifurcations, and ridge endings. (ii) An ideal system should be able to generate *full-finger impressions* as opposed to partial fingerprints. (iii) Synthetic fingerprints should be *computationally indistinguishable* from real impressions to be used as a means to extend the security of biometric storage systems. (iv) The system should be fully automated, requiring no manual feature engineering to have high *scalability*. As we show in the rest of this chapter, SynFi satisfies all of the above requirements.

Contributions. Our concrete contributions are as follows.

- We propose a new framework to generate robust full-finger synthetic fingerprints. We explore several deep learning-based solutions to generate high-quality samples. We formulate this task based on generative adversarial network and super-resolution methodologies.
- We perform qualitative as well as quantitative analysis on distinguishability of synthetic fingerprints from real ones using six different machine learning models.
- We provide the proof-of-concept implementation of our proposed methodology in Pytorch. We open-sourced our framework to facilitate progress, improvements, and verification process of fingerprint-based systems.

9.2 Prior Art

The prior work on generating synthetic fingerprints can be categorized into two broad groups. The first group is based on formulating mathematical models to generate artificial fingerprints. The second group leverages various classes of deep learning models. Generally speaking, the first group involves more feature engineering and manual tuning, whereas the second group inherits a more automated nature of feature extraction of the DL models.

Mathematical Models. One of the systems based on mathematical models is called SFinGe [CMM04]. In this system, generating a synthetic fingerprint involves four main phases: (i) a fingerprint shape is randomly generated via specific geometric models, (ii) a directional map is produced, (iii) a density map is created, and (iv) the first three maps are combined to generate a fingerprint pattern using a ridge-flow model. Finally, noise is added to make the generated image more realistic.

Unfortunately, solutions based on the mathematical models suffer from the low level of entropy due to the rigorous structure of the generation process. In contrast, SynFi generates each fingerprint starting from a completely *random noise*.

Deep Learning Models. Deep learning has demonstrated a breakthrough in several applications and domains. There are several categories of DL models. Two of which that are explored for the task of synthetic fingerprint generation are (i) Fully Visible Belief Networks (FVBN) such as PixelRNN [OKK16] that can produce one pixel at a time. Similar to Recurrent Neural Networks (RNN) that generate text, FVBNs can be used to create pixels of a fingerprint image. One drawback of these networks is that the final output can often be noisy. (ii) The second group is based on Variational Autoencoders (VAE). Compared to FVBN, VAE usually produces smoother images. Another line of work focuses on *MasterPrints*, which are real or synthetic fingerprint templates at *feature-level* [RMTR18] that can fool fingerprint-based authentication systems and authenticate the attacker as a legitimate user. The idea was later generalized to

DeepMasterPrints, which are synthetic fingerprints at *image-level* [BRT+18].

However, state-of-the-art DL-based methods can only generate *low-quality partial* fingerprints [EMB17, EB18, JEB18]. In contrast, as we compare in Section 9.4, SynFi generates full-finger fingerprints with significantly higher resolution due to a novel DL formulation.

9.3 Methodology

The main challenge in producing high-quality synthetic fingerprints is *estimating the probability distribution* of real fingerprints. Given the probability distribution, one can sample from this distribution to generate new fingerprints. However, obtaining such distribution is a very non-trivial task.

In SynFi, we rely on GANs to estimate the probability distribution of real fingerprints. Unfortunately, due to the small volume of the publicly available datasets, the GAN model cannot generalize well and produce realistic-looking samples. The prior art explores this approach. We also validate this idea and show that the generated samples by this approach are not acceptable (see Figure 9.1). As can be seen, these samples have deficient quality. They can easily be distinguished even without relying on sophisticated Machine Learning (ML) models. Therefore, in SynFi, we capture the problem of generating synthetic fingerprints as a *two-phase process*.

In the first phase, we rely on a GAN model to estimate the probability distribution of real fingerprints and create a low-quality image out of a randomly generated vector representing the *latent variable*. In the second phase, we train and use a *Super-Resolution (SR)* model to transform the low-quality image into a realistic, high-quality sample. In this phase, the details and texture of ridge endings and bifurcations within the fingerprints are embedded into the image.

In order to train both GAN and SR models, we need a dataset of real fingerprints. In practice, however, these datasets comprise fingerprint images that are not centered and have unnecessary auxiliary information around the fingerprint image. Therefore, we need to preprocess

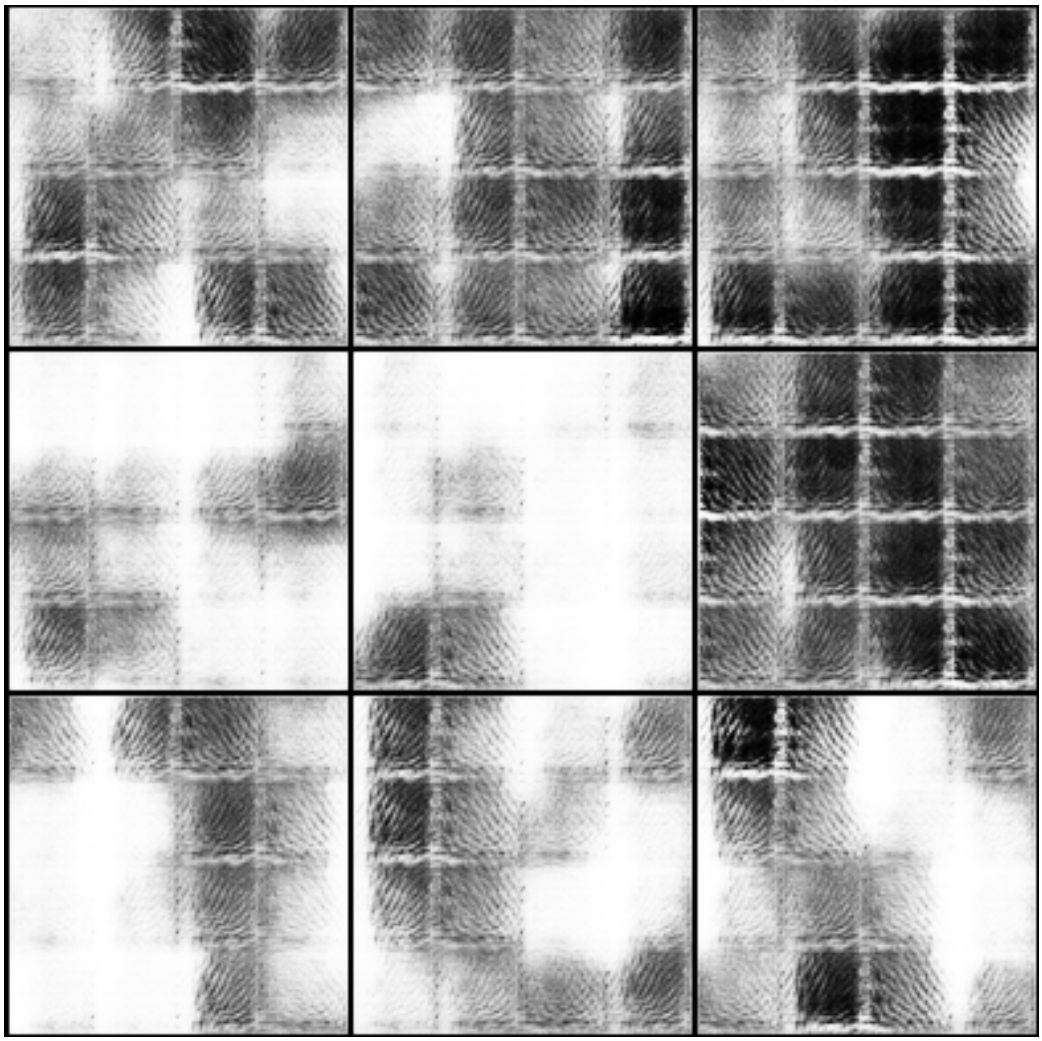


Figure 9.1: The result of generating 256×256 pixel images using GAN.

the dataset to enhance the quality of the images produced by both models. Figure 9.2 illustrates the overall design of SynFi and the relationship between different components. In what follows, we describe each of these components in more detail.

9.3.1 Pre-processing Phase: Fingerprint Segmentation

As can be seen in Figure 9.2, images in our dataset of real fingerprints contain some artifacts such as codes and numbers in the image. Besides, fingerprints are not centered. NIST biometric image software (NBIS) provides specific tools for processing fingerprint images. In this phase, we have to detect the boundary of each fingerprint within each image, crop, and scale

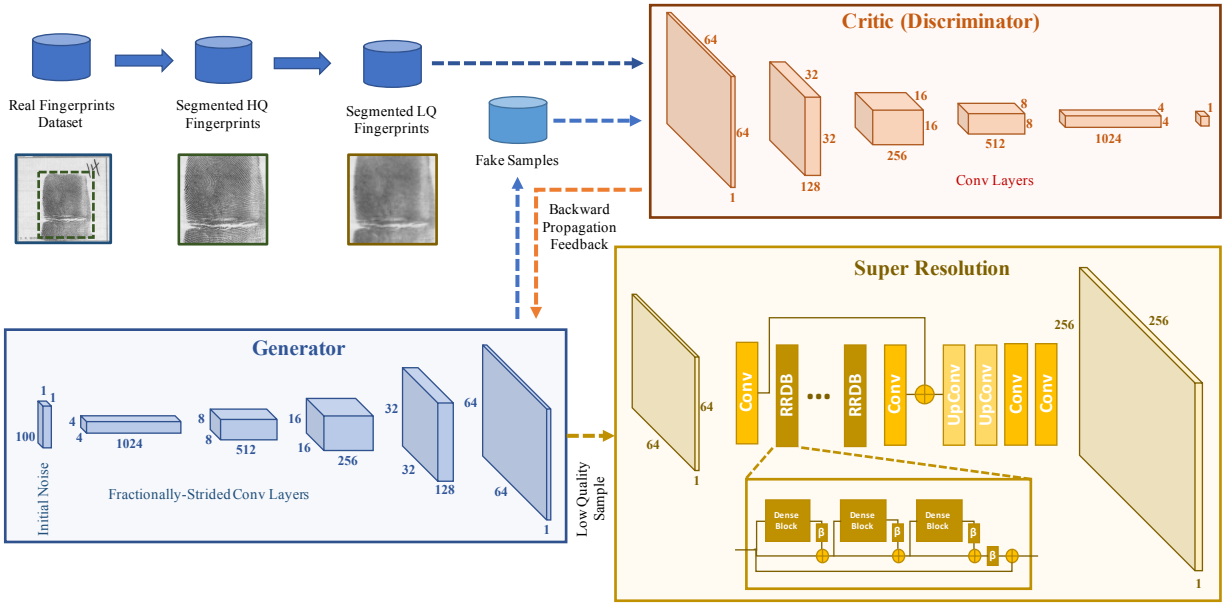


Figure 9.2: Overall design of SynFi for training (building the system) and execution (generating synthetic samples).

them accordingly. We observe that after this process, which we call *segmentation* in this work, the majority of the fingerprints have a resolution of 256×256 pixels. However, the GAN training procedure relies on much lower resolution images. Therefore, we create a *Low-Quality Database (LQD)* of 64×64 images, which can be used to train GAN. The SR model, on the other hand, needs a *High-Quality Database (HQD)* of 256×256 images in addition to LQD.

9.3.2 Phase 1: Generating Synthetic Fingerprints using GAN

After creating LQD, we train a GAN to generate (low-quality) synthetic fingerprints out of an input noise vector. There are dozen of different options for structures to use as our GAN network, in terms of the number and size of the layers and also the optimization loss and method used. However, due to the small volume of the real fingerprints that are publicly-available, there are two main obstacles for the GAN model to converge: *vanishing gradients* and *mode collapse*.

In this work, we choose Wasserstein GAN (WGAN) [ACB17] due to the following reasons. WGAN uses *Earth-Mover distance* as its loss function for comparing the target and real distributions. Unlike the traditional Minimax loss function, the Earth-Mover distance is a true

metric to measure distances in the space of probability distributions. This loss function helps the stabilization of the training process of GANs and reduces the possibility of several problems, including vanishing gradients and mode collapse.

The vanishing gradients problem arises because at the beginning of the training process, fake samples are easily distinguishable from the real samples; thus, the gradients computed during backpropagation are not helpful to tune the generative model. In WGAN, however, the discriminator model outputs a number instead of a probability estimation. The discriminator's job is to maximize the difference between the output number of real and fake samples (and not discriminate), thus, it is usually referred to as *critic*. This enables the gradients to be informative, even at the beginning of the training process.

The mode collapse problem is due to the fact that the generator can converge to a state that only produces a few plausible samples that can fool the discriminator. This problem is particularly important for us because our system has to have high entropy: generating many samples that are significantly different. Otherwise, SynFi cannot scale to generating millions or billions of unique samples. Relying on WGAN helps us to avoid the mode collapse problem since the discriminator can separately be trained to optimality and quickly detect fake samples, forcing the generator to search for new samples.

Even after incorporating the above optimizations and testing various configurations and different parameters, we observe that the trained GAN model is not capable of generating fingerprints with high quality for image sizes larger than 64×64 pixels. For instance, Figure 9.1 shows the output of GAN for 256×256 pixel samples. In order to produce high-quality images similar to publicly available datasets (256×256 pixel images), we need a second phase which we describe next.

9.3.3 Phase 2: Generalization to High-Quality Images

In the second phase of SynFi, the low-quality image is transformed into a high-resolution image with a more detailed texture. Improving the resolution and quality of images is one of the challenging and interesting problems in the Computer Vision community. Traditional super-resolution mechanisms improve the quality of the input image using many but lower-quality images. However, in our case, the low-quality image is generated from an initial noise, and we cannot produce multiple low-quality images of the *same* concept finger in Phase 1. Therefore, we have to explore *single-image super-resolution* solutions that take as input only a single low-resolution image and produce a higher-quality image. Single-image super-resolution is a significantly more challenging task. Fortunately, GANs help in this regard too. Recent advances in this area include but are not limited to [LTH+17, WYDL18, WYW+18].

After exploring several solutions in this area, we choose ESRGAN architecture with Residual-in-Residual Dense Blocks (RRDB). Details of our chosen architecture is provided in Figure 9.2 and Section 9.4. At a high-level, the architecture consists of a series of RRDBs surrounded by convolutional (Conv) layers. There exist upsampling layers after RRDBs and before the convolution layers. In contrast to Phase 1, we remove Batch Normalization (BN) layers, which is shown to enhance the quality of the produced images [LSK+17]. Adding BN layers increases the possibility of artifacts being added to the image. Relying on RRDB basic block enabled us to incorporate a higher number of hidden layers. We also leverage *residual scaling* [WYW+18]: residuals (the output of basic blocks) is scaled down by a constant ($0 < \beta \leqslant 1$) before being added to the main path. Additionally, we initialize the weights with small-variance random numbers to improve convergence.

In our SR model, the discriminator differs from Phase 1 in which it is a Relativistic average Discriminator (RaD) [JM18]. In Phase 1, the discriminator estimates the probability that a sample is real. In Phase 2, the discriminator estimates the probability that a real image is

(relatively) more realistic than a fake one. More precisely, RaD is formulated as:

$$D(x_r, x_f) = \sigma(C(x_r)) - \mathbb{E}_{x_f}[C(x_f)]$$

where $D(,)$ is the output of discriminator, x_r is the real sample, x_f is the fake sample, $C(.)$ is the non-transformed discriminator output, $\sigma(.)$ is the Sigmoid function, and $\mathbb{E}_{x_f}[.]$ represents taking an average over all fake samples in the batch. The discriminator's and generator's loss are:

$$L_D = -\mathbb{E}_{x_r}[\log(D(x_r, x_f))] - \mathbb{E}_{x_f}[\log(1 - D(x_f, x_r))]$$

$$L_G = -\mathbb{E}_{x_r}[\log(1 - D(x_r, x_f))] - \mathbb{E}_{x_f}[\log(D(x_f, x_r))]$$

After training the SR model using the above loss functions, the model is used to generate the final synthetic fingerprints, as depicted in Figure 9.2. The hyperparameters of the training process are described in Section 9.4.

9.4 Experimental Results

In this section, we first provide the details of our Computational setup, our dataset of real fingerprints, the training procedure, followed by comparison with the prior art. In the end, we provide extensive analysis of the indistinguishability of the SynFi's synthetic fingerprints from real ones.

9.4.1 Computational Environment

The experimental setup in which we train different components of SynFi as well as synthetic fingerprint generation phases is a server equipped with 128 GB of memory, two Intel Xeon E7 CPUs (12 core each), and four Nvidia Titan Xp GPUs (each with 12 GB of memory).

We develop the DL components in Pytorch[1].

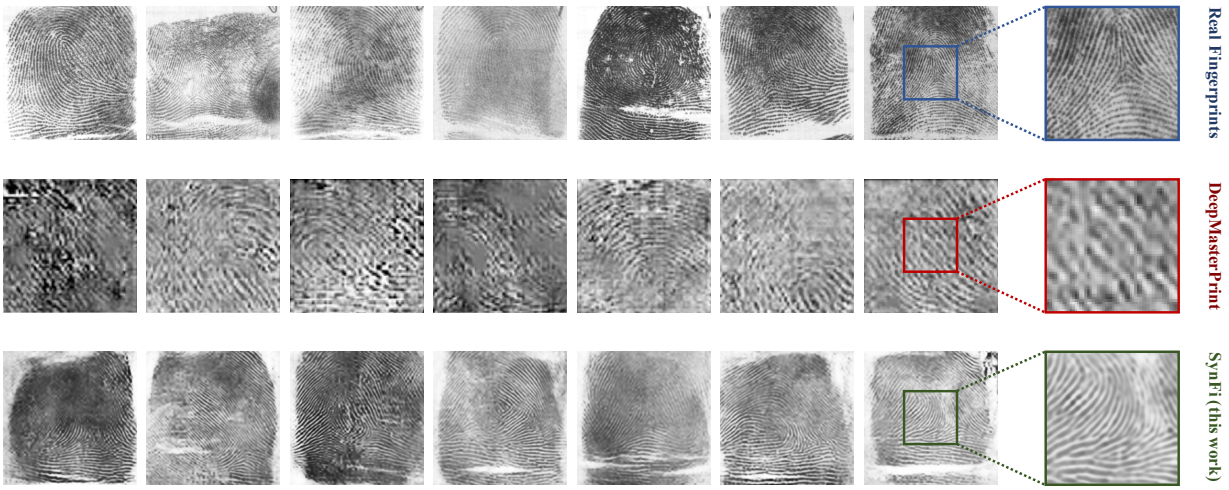


Figure 9.3: Quality comparison between real and synthetic fingerprint samples. *Top row:* NIST dataset real fingerprint samples. *Middle row:* Synthetic fingerprints generated by DeepMasterPrint [BRT+18]. *Bottom row:* Synthetic fingerprints generated by SynFi (this work).

Dataset. Our dataset of real fingerprints is the one provided by National Institute of Standards and Technology (NIST) in 2009, named Special Dataset (SD09)[2]. We have used this dataset to train both major components of SynFi: generative adversarial network and our super-resolution model. The NIST-SD09 dataset consists of 2700 subjects with all 10 fingerprint images. There are two impressions of each finger, resulting in 54000 fingerprint images overall. The data format is 8-bit gray-scale `png` images. There are additional metadata associated with each fingerprint, including the subject gender and the NCIC class [Kom05]: arch (A), left-loop (L), right-loop (R), tented-arch (T), whorl (W), and scar or mutilation (S). Most of the fingerprints belong to W, L, or R classes.

[1]Starting from the publicly available implementations at https://github.com/xinntao/BasicSR and https://github.com/martinarjovsky/WassersteinGAN

[2]https://www.nist.gov/srd/nist-special-database-9

9.4.2 Pre-processing Dataset

We use the `nfseq` tool provided by the NIST Biometric Image Software (NBIS) package to pre-process the real fingerprint dataset. As illustrated in Figure 9.2, this tool enables us to detect the precise boundary of the fingerprint and remove the unnecessary parts around the fingerprint itself. This step is crucial to enhance the quality of the images that are produced in both Phase 1 and Phase 2.

9.4.3 Architectures

The generator and critic components of Phase 1 have similar architecture but in a reversed order. The generator starts with a noise vector of size 100. Then the vector goes through a series of fractionally-strided convolutions in which the number of channels is reduced while the image size is increased, both by a factor of two. In the critic model, the intermediate layers are regular convolution layers.

The SR model has a more complex architecture. In the beginning, there is a convolutional layer followed by a series of 23 basic blocks. In the end, there are two upsampling and two convolution layers. Each basic block consists of three residual sub-blocks where each sub-block has five convolutional layers. The convolutional layers have 64 channels with a kernel size of 3. The activation function in RRDB is a leaky ReLU with a slope of 0.2 in the negative part. The output of the SR model is a 256×256 image with one channel (a gray-scale image).

9.4.4 Synthetic Samples and Qualitative Comparison

Figure 9.3 shows a set of samples of (i) real fingerprints in the NIST dataset, (ii) synthetic fingerprints generated by DeepMasterPrints [BRT+18], state-of-the-art DL-based method, and (iii) synthetic samples generated by our system. As can be seen from this figure, the output of SynFi is significantly more realistic compared to the prior art. Moreover, our methodology

can generate a full impression of fingerprints as opposed to the partial fingerprints generated by DeepMasterPrints. The rightmost column shows a magnified view of the details of the impressions, which shows the quality of the produced samples in SynFi. Next, we provide extensive experimental results to quantitatively compare SynFi samples with real fingerprints.

9.4.5 Indistinguishability and Quantitative Comparison

As we briefly discussed before, one of the most important characteristics of synthetic fingerprints is their *indistinguishability* from the real samples. Otherwise, not only synthetic samples cannot improve the quality and performance of authentication systems during development time, but also they cannot improve the security of storage systems for fingerprints as they can easily be distinguished and separated.

Figure 9.3 shows that the synthetic fingerprints generated by our system are visually very similar to the baseline NIST dataset of real fingerprints. However, to quantify how distinguishable synthetic fingerprints are from real ones, we perform the following analysis. We partition the subjects in the NIST dataset into training and test samples with 2200 and 500 subjects, respectively. Similarly, we create two disjoint sets of synthetic fingerprints, one for the training phase and one for the test phase. In order to minimize the classifier's bias, we put an equal number of real and synthetic fingerprints in the test dataset.

We train six different machine learning models: a Logistic Regression (LR) model, a Support Vector Machine (SVM) with linear kernel, a Random Forest with 10 estimators, and three different Deep Neural Network (DNN) models with four, five, and eight layers. The training process is formalized as a binary classification problem in which real fingerprints are labeled as zero, and synthetic samples are labeled as one. After training the six ML models, we evaluate them on an unseen test set consisting of real and synthetic samples.

The performance of these binary classifiers are reported in Table 9.1 using three standard metrics: *Accuracy (ACC)* which reflects the percentage of correct answers by the classifier, *False*

Positive Rate (FPR) which is defined as the ratio of wrongly classified samples as positive over all negative samples (both truly negative and incorrectly classified samples as positive). The third metric *False Negative Rate (FNR)* is the ratio of the number of samples falsely labeled as negative over all positive samples (both true positives and misclassified samples as negative). During the training process, these ML models were trained to learn the underlying pattern of fingerprints and reached the training accuracy of up to 99.76%. However, when evaluating these models on a set of unseen samples, the best performing classifier was the four-layer DNN with 100 and 20 neurons in the hidden layers and classification accuracy of **50.43%**. **In other words, the best classifier could distinguish synthetic fingerprints from real ones only 0.43% better than a random guess.**

Table 9.1: Analyzing indistinguishability of synthetic fingerprints against the real ones using various machine learning models.

Model Type	Model Description	ACC(%)	FPR(%)	FNR(%)
Logistic Regression	L2 regularization	49.99	**0.01**	99.99
Linear SVM	L2 regularization, C=1	50.01	0.06	99.91
Random Forest	Using 10 estimators	49.47	13.08	87.96
4-Layer DNN	Hidden Layers: 100, 20	**50.43**	19.78	79.35
5-Layer DNN	Hidden Layers: 100, 50, 10	50.35	11.36	87.93
8-Layer DNN	Hidden Layers: 800, 400, 200, 100, 50, 20	49.85	23.25	**77.03**

One can also analyze the effectiveness of the classifiers using a Receiver Operating Characteristic (ROC) curve. The ROC diagaram depicts True Positive Rate (TPR) against FPR. TPR is defined as $\text{TPR} = 1 - \text{FNR}$. Figure 9.4 shows the ROC curve of five ML models (ROC curve is not well-defined for SVMs). Relying on a purely random guess results in the black diagonal dashed line, which is the baseline for indistinguishability. Conceptually, ROC visualizes the fact that the classifiers' threshold for classifying an image as real or fake results in a trade-off between FPR and FNR (or TPR). Choosing a very low threshold leads to marking many real images as fake, hence, high FPR. Choosing a very high threshold results in outputting many fake

images as real, thus, high FNR. However, as can be seen, regardless of the chosen value for the threshold, none of the classifiers can perform reasonably better than a purely random guess.

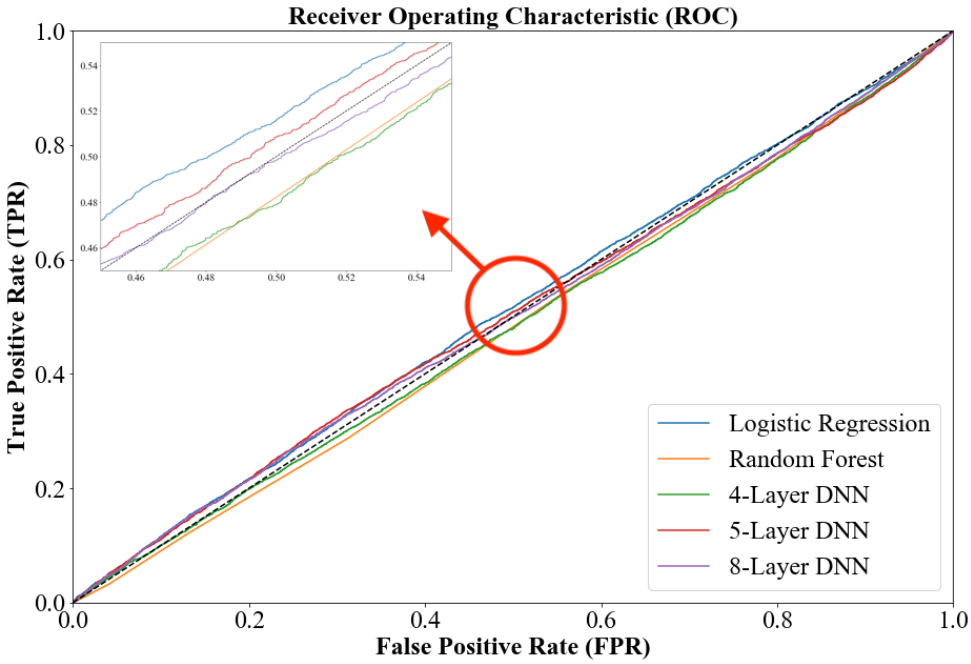


Figure 9.4: The ROC curve of five different machine learning models in distinguishing real fingerprints from synthetic ones generated by SynFi.

9.5 Summary

We present SynFi, an automated framework to generate large volume of high-quality synthetic fingerprints. We formulate this task as two disjoint and parallel learning problems to cope with the limited availability of real fingerprint samples. Our fingerprint generation data flow involves two phases: one based on generative adversarial network and one based on super-resolution methodologies. We perform extensive experiments and empirically show that our synthetic fingerprints inherit fine-grained texture of real samples such as ridge endings and bifurcations. Finally, we verify that the best performing machine learning model that we identified could distinguish synthetic fingerprints from real ones only 0.43% better than a random guess,

illustrating the effectiveness of SynFi to enhance the security of fingerprint storage systems.

9.6 Acknowledgements

This chapter, in part, has been submitted to the 2020 Design Automation Conference (DAC) as: M. Sadegh Riazi, Seyed Mohammad Chavoshian, Farinaz Koushanfar, "SynFi: Automatic Synthetic Fingerprint Generation" [RCK20]. The dissertation author was the primary author of this material.

Chapter 10

Future Research Directions

Secure computation is a promising technology that enables mutually distrusting parties to collaborate and creates a new platform for trust-free computations. This technology can, in turn, influence many industries and create new markets for data monetization while preserving the privacy of data owners. However, secure computation inherently has radically different computational cost models compared to plaintext computations. Basic operations such as random access or arithmetic operation have different computation costs in secure computation than in traditional plaintext executions. Several well-known algorithms that provide efficient (or even optimal) solutions for a given task do not necessarily keep their advantages in the secure computation realm. As a result, a new set of algorithms are needed for a variety of security and privacy-sensitive applications that are customized and designed for secure computation protocols.

In addition to algorithm-level research opportunities, different secure computation protocols and techniques can be customized and combined to yield more efficient computations. In general and in an ideal scenario, for a given task, the algorithm and the protocol can be co-designed to provide maximum efficiency at the cost of the longer design process. In the past few years and as has been shown in this thesis, mixed-protocol solutions can be a promising research direction for several tasks, including machine learning applications. Careful integration

of distinct secure computation protocols allows one to benefit from unique characteristics of each protocol.

Last but not least, new hardware platforms are increasingly needed to reduce the computational overhead of secure computation and bridge the gap between the plaintext executions and secure counterparts. Current ubiquitous hardware platforms, including CPUs and GPUs, are not explicitly designed for cryptographic operations and secure computation protocols. Thus, they are not necessarily well-suited for such tasks. In contrast, new hardware architecture and computation platforms can significantly increase the adoption of new secure computation technologies.

Appendix A

Network Network Architectures in XONN

Table A.1- A.7 present the network architectures for MNIST and CIFAR1-10 datasets, respectively. The notation of layer names is as follows: convolution $\rightarrow$ **CONV**, batch normalization $\rightarrow$ **BN**, binary activation $\rightarrow$ **BA**, max pooling $\rightarrow$ **MP**, and fully-connected $\rightarrow$ **FC**.

Table A.1: Evaluated neural network architectures on MNIST dataset.

BM1

1	**FC** [input: 784, output: $128s$] + **BN** + **BA**
2	**FC** [input: $128s$, output: $128s$] + **BN** + **BA**
3	**FC** [input: $128s$, output: 10] + **BN** + **Softmax**

BM2

1	**CONV** [input: $28 \times 28 \times 1$, window: 5×5, stride: 2, kernels: $5s$, output: $12 \times 12 \times 5s$] + **BN** + **BA**
2	**FC** [input: $720s$, output: $100s$] + **BN** + **BA**
3	**FC** [input: $100s$, output: 10] + **BN** + **Softmax**

BM3

1	**CONV** [input: $28 \times 28 \times 1$, window: 5×5, stride: 1, kernels: $16s$, output: $24 \times 24 \times 16s$] + **BN** + **BA**
2	**MP** [input: $24 \times 24 \times 16s$, window: 2×2, output: $12 \times 12 \times 16s$]
3	**CONV** [input: $12 \times 12 \times 16s$, window: 5×5, stride: 1, kernels: $16s$, output: $8 \times 8 \times 16s$] + **BN** + **BA**
4	**MP** [input: $8 \times 8 \times 16s$, window: 2×2, output: $4 \times 4 \times 16s$]
5	**FC** [input: $256s$, output: $100s$] + **BN** + **BA**
6	**FC** [input: $100s$, output: 10] + **BN** + **Softmax**

A.1 Acknowledgements

Appendix A, in part, has been published at the Proceedings of 2019 USENIX Security Symposium and appeared as M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, Farinaz Koushanfar, “XONN: XNOR-based Oblivious Deep Neural Network Inference”. The dissertation author was the primary author of this material.

Table A.2: Evaluated neural network architectures for CIFAR-10 dataset.

BC1

1	**CONV** [input: $32 \times 32 \times 3$, window: 3×3, stride: 1, kernels: $64s$, output: $30 \times 30 \times 64s$] + **BN** + **BA**
2	**CONV** [input: $30 \times 30 \times 64s$, window: 3×3, stride: 1, kernels: $64s$, output: $28 \times 28 \times 64s$] + **BN** + **BA**
3	**MP** [input: $28 \times 28 \times 64s$, window: 2×2, output: $14 \times 14 \times 64s$]
4	**CONV** [input: $14 \times 14 \times 64s$, window: 3×3, stride: 1, kernels: $64s$, output: $12 \times 12 \times 64s$] + **BN** + **BA**
5	**CONV** [input: $12 \times 12 \times 64s$, window: 3×3, stride: 1, kernels: $64s$, output: $10 \times 10 \times 64s$] + **BN** + **BA**
6	**MP** [input: $10 \times 10 \times 64s$, window: 2×2, output: $5 \times 5 \times 64s$]
7	**CONV** [input: $5 \times 5 \times 64s$, window: 3×3, stride: 1, kernels: $64s$, output: $3 \times 3 \times 64s$] + **BN** + **BA**
8	**CONV** [input: $3 \times 3 \times 64s$, window: 1×1, stride: 1, kernels: $64s$, output: $3 \times 3 \times 64s$] + **BN** + **BA**
9	**CONV** [input: $3 \times 3 \times 64s$, window: 1×1, stride: 1, kernels: $16s$, output: $3 \times 3 \times 16s$] + **BN** + **BA**
10	**FC** [input: $144s$, output: 10] + **BN** + **Softmax**

BC2

1	**CONV** [input: $32 \times 32 \times 3$, window: 3×3, stride: 1, kernels: $16s$, output: $32 \times 32 \times 16s$] + **BN** + **BA**
2	**CONV** [input: $32 \times 32 \times 16s$, window: 3×3, stride: 1, kernels: $16s$, output: $32 \times 32 \times 16s$] + **BN** + **BA**
3	**CONV** [input: $32 \times 32 \times 16s$, window: 3×3, stride: 1, kernels: $16s$, output: $32 \times 32 \times 16s$] + **BN** + **BA**
4	**MP** [input: $32 \times 32 \times 16s$, window: 2×2, output: $16 \times 16 \times 16s$]
5	**CONV** [input: $16 \times 16 \times 16s$, window: 3×3, stride: 1, kernels: $32s$, output: $16 \times 16 \times 32s$] + **BN** + **BA**
6	**CONV** [input: $16 \times 16 \times 32s$, window: 3×3, stride: 1, kernels: $32s$, output: $16 \times 16 \times 32s$] + **BN** + **BA**
7	**CONV** [input: $16 \times 16 \times 32s$, window: 3×3, stride: 1, kernels: $32s$, output: $16 \times 16 \times 32s$] + **BN** + **BA**
8	**MP** [input: $16 \times 16 \times 32s$, window: 2×2, output: $8 \times 8 \times 32s$]
9	**CONV** [input: $8 \times 8 \times 32s$, window: 3×3, stride: 1, kernels: $48s$, output: $6 \times 6 \times 48s$] + **BN** + **BA**
10	**CONV** [input: $6 \times 6 \times 48s$, window: 3×3, stride: 1, kernels: $48s$, output: $4 \times 4 \times 48s$] + **BN** + **BA**
11	**CONV** [input: $4 \times 4 \times 48s$, window: 3×3, stride: 1, kernels: $64s$, output: $2 \times 2 \times 64s$] + **BN** + **BA**
12	**MP** [input: $2 \times 2 \times 64s$, window: 2×2, output: $1 \times 1 \times 64s$]
13	**FC** [input: $64s$, output: 10] + **BN** + **Softmax**

Table A.3: Evaluated neural network architectures for CIFAR-10 dataset (BC3 and BC4 networks).

BC3

1	**CONV** [input: $32 \times 32 \times 3$, window: 3×3, stride: 1, kernels: $16s$, output: $32 \times 32 \times 16s$] + **BN** + **BA**
2	**CONV** [input: $32 \times 32 \times 16s$, window: 3×3, stride: 1, kernels: $32s$, output: $32 \times 32 \times 32s$] + **BN** + **BA**
3	**CONV** [input: $32 \times 32 \times 32s$, window: 3×3, stride: 1, kernels: $32s$, output: $32 \times 32 \times 32s$] + **BN** + **BA**
4	**MP** [input: $32 \times 32 \times 32s$, window: 2×2, output: $16 \times 16 \times 32s$]
5	**CONV** [input: $16 \times 16 \times 32s$, window: 3×3, stride: 1, kernels: $48s$, output: $16 \times 16 \times 48s$] + **BN** + **BA**
6	**CONV** [input: $16 \times 16 \times 48s$, window: 3×3, stride: 1, kernels: $64s$, output: $16 \times 16 \times 64s$] + **BN** + **BA**
7	**CONV** [input: $16 \times 16 \times 64s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
8	**MP** [input: $16 \times 16 \times 80s$, window: 2×2, output: $8 \times 8 \times 80s$]
9	**CONV** [input: $8 \times 8 \times 80s$, window: 3×3, stride: 1, kernels: $96s$, output: $6 \times 6 \times 96s$] + **BN** + **BA**
10	**CONV** [input: $6 \times 6 \times 96s$, window: 3×3, stride: 1, kernels: $96s$, output: $4 \times 4 \times 96s$] + **BN** + **BA**
11	**CONV** [input: $4 \times 4 \times 96s$, window: 3×3, stride: 1, kernels: $128s$, output: $2 \times 2 \times 128s$] + **BN** + **BA**
12	**MP** [input: $2 \times 2 \times 128s$, window: 2×2, output: $1 \times 1 \times 128s$]
13	**FC** [input: $128s$, output: 10] + **BN** + **Softmax**

Table A.4: Evaluated neural network architectures for CIFAR-10 dataset (BC4 network).

BC4	
1	**CONV** [input: $32 \times 32 \times 3$, window: 3×3, stride: 1, kernels: $32s$, output: $32 \times 32 \times 32s$] + **BN** + **BA**
2	**CONV** [input: $32 \times 32 \times 32s$, window: 3×3, stride: 1, kernels: $48s$, output: $32 \times 32 \times 48s$] + **BN** + **BA**
3	**CONV** [input: $32 \times 32 \times 48s$, window: 3×3, stride: 1, kernels: $64s$, output: $32 \times 32 \times 64s$] + **BN** + **BA**
4	**CONV** [input: $32 \times 32 \times 64s$, window: 3×3, stride: 1, kernels: $64s$, output: $32 \times 32 \times 64s$] + **BN** + **BA**
5	**MP** [input: $32 \times 32 \times 64s$, window: 2×2, output: $16 \times 16 \times 64s$]
6	**CONV** [input: $16 \times 16 \times 64s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
7	**CONV** [input: $16 \times 16 \times 80s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
8	**CONV** [input: $16 \times 16 \times 80s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
9	**CONV** [input: $16 \times 16 \times 80s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
10	**MP** [input: $16 \times 16 \times 80s$, window: 2×2, output: $8 \times 8 \times 80s$]
11	**CONV** [input: $8 \times 8 \times 80s$, window: 3×3, stride: 1, kernels: $128s$, output: $6 \times 6 \times 128s$] + **BN** + **BA**
12	**CONV** [input: $6 \times 6 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $4 \times 4 \times 128s$] + **BN** + **BA**
13	**CONV** [input: $4 \times 4 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $2 \times 2 \times 128s$] + **BN** + **BA**
14	**MP** [input: $2 \times 2 \times 128s$, window: 2×2, output: $1 \times 1 \times 128s$]
15	**FC** [input: $128s$, output: 10] + **BN** + **Softmax**

Table A.5: Evaluated neural network architectures for CIFAR-10 dataset (BC5 network).

	BC5
1	**CONV** [input: $32 \times 32 \times 3$, window: 3×3, stride: 1, kernels: $32s$, output: $32 \times 32 \times 32s$] + **BN** + **BA**
2	**CONV** [input: $32 \times 32 \times 32s$, window: 3×3, stride: 1, kernels: $32s$, output: $32 \times 32 \times 32s$] + **BN** + **BA**
3	**CONV** [input: $32 \times 32 \times 32s$, window: 3×3, stride: 1, kernels: $32s$, output: $32 \times 32 \times 32s$] + **BN** + **BA**
4	**CONV** [input: $32 \times 32 \times 32s$, window: 3×3, stride: 1, kernels: $48s$, output: $32 \times 32 \times 48s$] + **BN** + **BA**
5	**CONV** [input: $32 \times 32 \times 48s$, window: 3×3, stride: 1, kernels: $48s$, output: $32 \times 32 \times 48s$] + **BN** + **BA**
6	**MP** [input: $32 \times 32 \times 48s$, window: 2×2, output: $16 \times 16 \times 48s$]
7	**CONV** [input: $16 \times 16 \times 48s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
8	**CONV** [input: $16 \times 16 \times 80s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
9	**CONV** [input: $16 \times 16 \times 80s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
10	**CONV** [input: $16 \times 16 \times 80s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
11	**CONV** [input: $16 \times 16 \times 80s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
12	**CONV** [input: $16 \times 16 \times 80s$, window: 3×3, stride: 1, kernels: $80s$, output: $16 \times 16 \times 80s$] + **BN** + **BA**
13	**MP** [input: $16 \times 16 \times 80s$, window: 2×2, output: $8 \times 8 \times 80s$]
14	**CONV** [input: $8 \times 8 \times 80s$, window: 3×3, stride: 1, kernels: $128s$, output: $8 \times 8 \times 128s$] + **BN** + **BA**
15	**CONV** [input: $8 \times 8 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $8 \times 8 \times 128s$] + **BN** + **BA**
16	**CONV** [input: $8 \times 8 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $8 \times 8 \times 128s$] + **BN** + **BA**
17	**CONV** [input: $8 \times 8 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $6 \times 6 \times 128s$] + **BN** + **BA**
18	**CONV** [input: $6 \times 6 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $4 \times 4 \times 128s$] + **BN** + **BA**
19	**CONV** [input: $4 \times 4 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $2 \times 2 \times 128s$] + **BN** + **BA**
20	**MP** [input: $2 \times 2 \times 128s$, window: 2×2, output: $1 \times 1 \times 128s$]
21	**FC** [input: $128s$, output: 10] + **BN** + **Softmax**

Table A.6: Evaluated neural network architectures for CIFAR-10 dataset (BC6 network).

	BC6
1	**CONV** [input: $32 \times 32 \times 3$, window: 3×3, stride: 1, kernels: $16s$, output: $32 \times 32 \times 16s$] + **BN** + **BA**
2	**CONV** [input: $32 \times 32 \times 16s$, window: 3×3, stride: 1, kernels: $16s$, output: $32 \times 32 \times 16s$] + **BN** + **BA**
3	**MP** [input: $32 \times 32 \times 16s$, window: 2×2, output: $16 \times 16 \times 16s$]
4	**CONV** [input: $16 \times 16 \times 16s$, window: 3×3, stride: 1, kernels: $32s$, output: $16 \times 16 \times 32s$] + **BN** + **BA**
5	**CONV** [input: $16 \times 16 \times 32s$, window: 3×3, stride: 1, kernels: $32s$, output: $16 \times 16 \times 32s$] + **BN** + **BA**
6	**MP** [input: $16 \times 16 \times 32s$, window: 2×2, output: $8 \times 8 \times 32s$]
7	**CONV** [input: $8 \times 8 \times 32s$, window: 3×3, stride: 1, kernels: $64s$, output: $8 \times 8 \times 64s$] + **BN** + **BA**
8	**CONV** [input: $8 \times 8 \times 64s$, window: 3×3, stride: 1, kernels: $64s$, output: $8 \times 8 \times 64s$] + **BN** + **BA**
9	**CONV** [input: $8 \times 8 \times 64s$, window: 3×3, stride: 1, kernels: $64s$, output: $8 \times 8 \times 64s$] + **BN** + **BA**
10	**MP** [input: $8 \times 8 \times 64s$, window: 2×2, output: $4 \times 4 \times 64s$]
11	**CONV** [input: $4 \times 4 \times 64s$, window: 3×3, stride: 1, kernels: $128s$, output: $4 \times 4 \times 128s$] + **BN** + **BA**
12	**CONV** [input: $4 \times 4 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $4 \times 4 \times 128s$] + **BN** + **BA**
13	**CONV** [input: $4 \times 4 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $4 \times 4 \times 128s$] + **BN** + **BA**
14	**MP** [input: $4 \times 4 \times 128s$, window: 2×2, output: $2 \times 2 \times 128s$]
15	**CONV** [input: $2 \times 2 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $2 \times 2 \times 128s$] + **BN** + **BA**
16	**CONV** [input: $2 \times 2 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $2 \times 2 \times 128s$] + **BN** + **BA**
17	**CONV** [input: $2 \times 2 \times 128s$, window: 3×3, stride: 1, kernels: $128s$, output: $2 \times 2 \times 128s$] + **BN** + **BA**
18	**MP** [input: $2 \times 2 \times 128s$, window: 2×2, output: $1 \times 1 \times 128s$]
19	**FC** [input: $128s$, output: $512s$] + **BN** + **BA**
20	**FC** [input: $512s$, output: $512s$] + **BN** + **BA**
21	**FC** [input: $512s$, output: 10] + **BN** + **Softmax**

Table A.7: Evaluated neural network architectures for medical datasets.

BH1

1	**FC** [input: 30, output: 16] + **BN** + **BA**
2	**FC** [input: 16, output: 16] + **BN** + **BA**
3	**FC** [input: 16, output: 2] + **BN** + **Softmax**

BH2

1	**FC** [input: 8, output: 20] + **BN** + **BA**
2	**FC** [input: 20, output: 20] + **BN** + **BA**
3	**FC** [input: 20, output: 2] + **BN** + **Softmax**

BH3

1	**FC** [input: 10, output: 32] + **BN** + **BA**
2	**FC** [input: 32, output: 32] + **BN** + **BA**
3	**FC** [input: 32, output: 2] + **BN** + **Softmax**

BH4

1	**CONV** [input: $32 \times 32 \times 3$, window: 5×5, stride: 1, kernels: 36, output: $28 \times 28 \times 36$] + **BN** + **BA**
2	**MP** [input: $28 \times 28 \times 36$, window: 2×2, output: $14 \times 14 \times 36$]
3	**CONV** [input: $14 \times 14 \times 36$, window: 5×5, stride: 1, kernels: 36, output: $10 \times 10 \times 36$] + **BN** + **BA**
4	**MP** [input: $10 \times 10 \times 36$, window: 2×2, output: $5 \times 5 \times 36$]
5	**FC** [input: 900, output: 72] + **BN** + **BA**
6	**FC** [input: 72, output: 2] + **BN** + **Softmax**

Bibliography

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Operating Systems Design and Implementation (OSDI)*, 2016.

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 153–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[ABF+17] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *IEEE S&P*, 2017.

[ABVMA18] Ahmad Al Badawi, Bharadwaj Veeravalli, Chan Fook Mun, and Khin Mi Mi Aung. High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 70–95, 2018.

[ABZS13] Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure computation on floating point numbers. In *NDSS*, 2013.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[ACS03] Igor Arsovski, Trevis Chandler, and Ali Sheikholeslami. A ternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme. *IEEE Journal of Solid-State Circuits*, 2003.

[ADWF15] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831, 2015.

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *CCS*, 2016.

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, 2012.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*, 2013.

[Baa99] Bevan M Baas. *An approach to low-power, high-performance, fast Fourier transform processor design*. PhD thesis, Citeseer, 1999.

[Baa05] Bevan M Baas. A generalized cached-FFT algorithm. In *Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, volume 5, pages v–89. IEEE, 2005.

[Bar87] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[BBB+17] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. Secure multiparty computation from SGX. In *FC*, 2017.

[BBC+10] Mauro Barni, Tiziano Bianchi, Dario Catalano, Mario Di Raimondo, Ruggero Donida Labati, Pierluigi Failla, Dario Fiore, Riccardo Lazzeretti, Vincenzo Piuri, Fabio Scotti, and Alessandro Piva. Privacy-preserving fingercode authentication. In *Proceedings of the ACM workshop on Multimedia and security*, 2010.

[BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2009.

[BDJ+06] Peter Bogetoft, Ivan Damgård, Thomas P Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. A practical implementation of secure auctions based on multiparty integer computation. In *Financial Cryptography*. Springer, 2006.

[BDK+18] Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. HyCC: Compilation of hybrid protocols for practical secure computation. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 847–861. ACM, 2018.

[BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *CCS*, 2008.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In *CRYPTO*, 1995.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC*, 1996.

[BEDM+12] Joshua Baron, Karim El Defrawy, Kirill Minkovich, Rafail Ostrovsky, and Eric Tressler. 5pm: Secure pattern matching. In *International Conference on Security and Cryptography for Networks*. Springer, 2012.

[BEHZ16] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016: 23rd Annual International Workshop on Selected Areas in Cryptography*, volume 10532 of *Lecture Notes in Computer Science*, pages 423–442, St. John's, NL, Canada, August 10–12, 2016. Springer, Heidelberg, Germany.

[BELO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the Internet. In *CCS*, 2016.

[BFK+09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *ESORICS*, 2009.

[BFL+11] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-preserving ECG classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security*, 6(2):452–468, 2011.

[BG11] Marina Blanton and Paolo Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*, 2011.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE S&P*, 2013.

[BHWK16] Niklas Büscher, Andreas Holzer, Alina Weber, and Stefan Katzenbeisser. Compiling low depth circuits for practical secure computation. In *ESORICS*, 2016.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM CCS*, 2017.

[BK15a] Elaine Barker and John Kelsey. NIST special publication 800-90a revision 1: Recommendation for random number generation using deterministic random bit generators. Technical report, 2015.

[BK15b] Niklas Büscher and Stefan Katzenbeisser. Faster secure computation through automatic parallelization. In *USENIX Security*, 2015.

[BKJK16] Niklas Büscher, David Kretzmer, Arnav Jindal, and Stefan Katzenbeisser. Scalable secure computation from ansi-c. In *WIFS*, 2016.

[BLLN13] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In Martijn Stam, editor, *14th IMA International Conference on Cryptography and Coding*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64, Oxford, UK, December 17–19, 2013. Springer, Heidelberg, Germany.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, 2008.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM, 1990.

[BNTW12] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6), 2012.

[BOP06] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, pages 146–151. ACM, 2006.

[BPA+19] Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Kurt Rohloff. Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2019.

[BPSW07] Justin Brickell, Donald E Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *CCS*, 2007.

[BR11] Petros Boufounos and Shantanu Rane. Secure binary embeddings for privacy preserving nearest neighbors. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2011.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in cryptology–crypto 2012*, pages 868–886. Springer, 2012.

[bre] Breast Cancer Wisconsin, accessed on 01/20/2019. `https://www.kaggle.com/uciml/breast-cancer-wisconsin-data`.

[BRT+18] Philip Bontrager, Aditi Roy, Julian Togelius, Nasir Memon, and Arun Ross. DeepMasterPrints: Generating masterprints for dictionary attacks via latent variable evolution. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–9. IEEE, 2018.

[BSMD10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. *USENIX Security*, 2010.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[BVMA18] Ahmad Al Badawi, Bharadwaj Veeravalli, Chan Fook Mun, and Khin Mi Mi Aung. High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2):70–95, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/875`.

[CBL+18] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster CryptoNets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

[CCD+18] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M Sadegh Riazi. Scaling up secure nearest neighbor search. *NeurIPS Workshop on Privacy-Preserving Machine Learning*, 2018.

[CCD+19] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M Sadegh Riazi. SANNS: Scaling up secure approximate k-nearest neighbors search. *arXiv preprint arXiv:1904.02033*, 2019.

[CCM08] Michael R Clarkson, Stephen Chong, and Andrew C Myers. Civitas: Toward a secure voting system. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 354–368. IEEE, 2008.

[CDKS19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. Cryptology ePrint Archive, Report 2019/524, 2019. `https://eprint.iacr.org/2019/524`.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[CGR+17] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. EzPC: Programmable, efficient, and scalable secure two-party computation. Cryptology ePrint Archive, Report 2017/1109, 2017.

[CGRS14] David Bruce Cousins, John Golusky, Kurt Rohloff, and Daniel Sumorok. An FPGA co-processor implementation of homomorphic encryption. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2014.

[CHK+12] Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *Cryptographers' Track at the RSA Conference*, pages 416–432. Springer, 2012.

[CHK+19] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In Carlos Cid and Michael J. Jacobson Jr:, editors, *SAC 2018: 25th Annual International Workshop on Selected Areas in Cryptography*, volume 11349 of *Lecture Notes in Computer Science*, pages 347–368, Calgary, AB, Canada, August 15–17, 2019. Springer, Heidelberg, Germany.

[Cho15] François Chollet. Keras. https://keras.io, 2015.

[CHS+16] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[CLT14] Henry Carter, Charles Lever, and Patrick Traynor. Whitewash: outsourcing garbled circuit generation for mobile devices. In *ACSAC*, 2014.

[CMM04] Raffaele Cappelli, D Maio, and D Maltoni. Sfinge: an approach to synthetic fingerprint generation. In *International Workshop on Biometric Technologies (BT2004)*, pages 147–154, 2004.

[CMO+13] Xiaolin Cao, Ciara Moore, Máire O'Neill, Elizabeth O'Sullivan, and Neil Hanley. Accelerating fully homomorphic encryption over the integers with super-size hardware multiplier and modular reduction. *IACR Cryptology ePrint Archive*, 2013:616, 2013.

[CMO+14] Xiaolin Cao, Ciara Moore, Máire O'Neill, Neil Hanley, and Elizabeth O'Sullivan. High-speed fully homomorphic encryption over the integers. In *International Conference on Financial Cryptography and Data Security*, pages 169–180. Springer, 2014.

[CMTB13] Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Secure outsourced garbled circuit evaluation for mobile devices. In *USENIX Security*, 2013.

[CMTB15] Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Outsourcing secure two-party computation as a black box. In *CANS*, 2015.

[CMTB16] Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Secure outsourced garbled circuit evaluation for mobile devices. In *Journal of Computer Security*, 2016.

[com17a] Mu9c1480b datasheet. http://www.datasheetcatalog.com/datasheets_pdf/M/U/9/C/MU9C1480B-50TAC.shtml, 2017.

[com17b] Mu9c4320l datasheet. http://pdf1.alldatasheet.com/datasheet-pdf/view/150649/MUSIC/MU9C4320L.html, 2017.

[CRC+19] Rahul Chatterjee, M Sadegh Riazi, Tanmoy Chowdhury, Emanuela Marasco, Farinaz Koushanfar, and Ari Juels. Multisketches: Practical secure sketches using off-the-shelf biometric matching algorithms. In *ACM CCS*, 2019.

[CRS16] David Bruce Cousins, Kurt Rohloff, and Daniel Sumorok. Designing an FPGA-accelerated homomorphic encryption co-processor. *IEEE Transactions on Emerging Topics in Computing*, 5(2):193–206, 2016.

[CS10] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *FC*, 2010.

[CT12] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[cuF] cuFHE. `https://github.com/vernamlab/cuFHE`. Vernam Group.

[CW08] Emmanuel J Candès and Michael B Wakin. An introduction to compressive sampling. *IEEE signal processing magazine*, 2008.

[DA01] Wenliang Du and Mikhail J Atallah. Protocols for secure remote database access with approximate matching. In *E-Commerce Security and Privacy*, 2001.

[DCFT13] Emiliano De Cristofaro, Sky Faber, and Gene Tsudik. Secure genomic testing with size-and position-hiding private substring matching. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. ACM, 2013.

[DCT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography*, volume 10, pages 143–159. Springer, 2010.

[DDK+15] Daniel Demmler, Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. Automated synthesis of optimized circuits for secure computation. In *CCS*, 2015.

[DDS14] Wei Dai, Yarkın Doröz, and Berk Sunar. Accelerating NTRU based homomorphic encryption using GPUs. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2014.

[DES16] Jack Doerner, David Evans, and Abhi Shelat. Secure stable matching at scale. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1602–1613. ACM, 2016.

[DGBL+16a] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, 2016.

[DGBL+16b] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, 2016.

[DGKN09] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *PKC*, 2009.

[dia] Pima Indians Diabetes, accessed on 01/20/2019. `https://www.kaggle.com/uciml/pima-indians-diabetes-database`.

[DÖS13] Yarkin Doröz, Erdinç Öztürk, and Berk Sunar. Evaluating the hardware performance of a million-bit multiplier. In *2013 Euromicro Conference on Digital System Design*, pages 955–962. IEEE, 2013.

[DÖS14a] Yarkın Doröz, Erdinç Öztürk, and Berk Sunar. Accelerating fully homomorphic encryption in hardware. *IEEE Transactions on Computers*, 64(6):1509–1521, 2014.

[DÖS14b] Yarkın Doröz, Erdinç Öztürk, and Berk Sunar. A million-bit multiplier architecture for fully homomorphic encryption. *Microprocessors and Microsystems*, 38(8):766–775, 2014.

[DÖSS15] Yarkın Doröz, Erdinç Öztürk, Erkay Savaş, and Berk Sunar. Accelerating LTV based homomorphic encryption in reconfigurable hardware. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 185–204. Springer, 2015.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.

[DR13] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[DS16] Wei Dai and Berk Sunar. cuHE: A homomorphic encryption accelerator library. In Enes Pasalic and Lars R. Knudsen, editors, *Cryptography and Information Security in the Balkans*, pages 169–186, Cham, 2016. Springer International Publishing.

[DSC+18] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Chet: Compiler and runtime for homomorphic evaluation of tensor programs. *arXiv preprint arXiv:1810.00845*, 2018.

[DSZ14] Daniel Demmler, Thomas Schneider, and Michael Zohner. Ad-Hoc secure two-party computation on mobile devices using hardware tokens. In *USENIX Security*, 2014.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[DWC10] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. Ieee, 2010.

[Dwo08] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*. Springer, 2008.

[EB18] Nabil Ettehadi and Aman Behal. Implementation of feeding task via learning from demonstration. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pages 274–277. IEEE, 2018.

[EBVL11] Zekeriya Erkin, Michael Beye, Thijs Veugen, and Reginald L Lagendijk. Efficiently computing private recommendations. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011.

[EFG+09] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *PETS*, 2009.

[EKN+17] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[EMB17] N Ettehadi, S Manaffam, and A Behal. Learning from demonstration: Generalization via task segmentation. In *IOP Conference Series: Materials Science and Engineering*, volume 261, page 012001. IOP Publishing, 2017.

[ERR+19] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24, 2019.

[FIKS06] Raymond Fisman, Sheena S Iyengar, Emir Kamenica, and Itamar Simonson. Gender differences in mate selection: Evidence from a speed dating experiment. *The Quarterly Journal of Economics*, 2006.

[FIL17] Xin Fang, Stratis Ioannidis, and Miriam Leeser. Secure function evaluation using an FPGA overlay architecture. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 257–266. ACM, 2017.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM CCS*. ACM, 2015.

[FLNW17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT*, 2017.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251. Springer, 1992.

[FPRSJ04] Joan Feigenbaum, Benny Pinkas, Raphael Ryger, and Felipe Saint-Jean. Secure computation of surveys. In *EU Workshop on Secure Multiparty Protocols*, 2004.

[Fri09] Keith B Frikken. Practical private DNA string searching and matching through efficient oblivious automata evaluation. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2009.

[FV-] FV-NFLlib. `https://github.com/CryptoExperts/FV-NFLlib`. CryptoExperts.

[FV12a] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[FV12b] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `http://eprint.iacr.org/2012/144`.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[GG10] Ashish Goel and Pankaj Gupta. Small subset queries and bloom filters using ternary associative memories, with applications. *ACM SIGMETRICS Performance Evaluation Review*, 2010.

[GH11] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 129–148. Springer, 2011.

[GMG+13] Melissa Gymrek, Amy L McGuire, David Golan, Eran Halperin, and Yaniv Erlich. Identifying personal genomes by surname inference. *Science*, 339(6117), 2013.

[GMGM13] Mohammed Gharib, Mohsen Minaei, Morteza Golkari, and Ali Movaghar. Expert key selection impact on the manets' performance using probabilistic key management algorithm. In *Proceedings of the 6th International Conference on Security of Information and Networks*, pages 347–351, 2013.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, 1987.

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[goo17] Google fires engineer for privacy breach. `https://www.cnet.com/news/google-fired-engineer-for-privacy-breach/`, 2017.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, 2014.

[GS62] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 1962.

[GS06] J Raphael Gibbs and Andrew Singleton. Application of genome-wide single nucleotide polymorphism typing: simple association and beyond. *PLoS Genet*, 2006.

[GSK18] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. ReBNet: Residual binarized neural network. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 57–64. IEEE, 2018.

[HAPC17] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In *ACM CCS*, 2017.

[HB18] Shakirah Hashim and Mohammed Benaissa. Accelerating integer based fully homomorphic encryption using frequency domain multiplication. In *International Conference on Information and Communications Security*, pages 161–176. Springer, 2018.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security*, 2011.

[HFKV12] Andreas Holzer, Martin Franz, Stefan Katzenbeisser, and Helmut Veith. Secure two-party computations in ANSI C. In *CCS*, 2012.

[HK19] Siam U Hussain and Farinaz Koushanfar. FASE: FPGA acceleration of secure function evaluation. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 280–288. IEEE, 2019.

[HKE13] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *Advances in Cryptology–CRYPTO 2013*, pages 18–35. Springer, 2013.

[HKS+10] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *CCS*, 2010.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography Conference*. Springer, 2008.

[HN08] Jay Heiser and Mark Nicolett. Assessing the security risks of cloud computing. *Gartner report*, 27:29–52, 2008.

[HPS19] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, volume 11405 of *Lecture Notes in Computer Science*, pages 83–105, San Francisco, CA, USA, March 4–8, 2019. Springer, Heidelberg, Germany.

[HPTD15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[HRGK18] Siam U Hussain, Bita Darvish Rouhani, Mohammad Ghasemzadeh, and Farinaz Koushanfar. MAXelerator: FPGA accelerator for privacy preserving multiply-accumulate (MAC) on cloud servers. In *Proceedings of the 55th Annual Design Automation Conference*, page 33. ACM, 2018.

[HRK18] Siam Umar Hussain, M Sadegh Riazi, and Farinaz Koushanfar. SHAIP: Secure hamming distance for authentication of intrinsic pufs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(6):1–20, 2018.

[HRK20] Siam U Hussain, M Sadegh Riazi, and Farinaz Koushanfar. The fusion of secure function evaluation and logic synthesis. *IEEE Security and Privacy (S&P) Magazine.*, 2020.

[HS12] Amir Herzberg and Haya Shulman. Oblivious and fair server-aided two-party computation. In *ARES*, 2012.

[HSR+08] 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 Genet*, 4(8), 2008.

[HT10] Carmit Hazay and Tomas Toft. Computationally secure pattern matching in the presence of malicious adversaries. In *ASIACRYPT*, 2010.

[HTGW18] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies*, 2018(3):123–142, 2018.

[Hua12] Yan Huang. *Practical Secure Two-Party Computation*. PhD thesis, University of Virginia, 2012.

[Hua16] Zhiyi Huang. Privacy preserving auction., 2016.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, 2003.

[IKR+19] Mohsen Imani, Yeseong Kim, Sadegh Riazi, John Messerly, Patric Liu, Farinaz Koushanfar, and Tajana Rosing. A framework for collaborative learning in secure high-dimensional space. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 435–446. IEEE, 2019.

[IKRR16] Mohsen Imani, Yeseong Kim, Abbas Rahimi, and Tajana Rosing. ACAM: Approximate computing based on adaptive associative memory with online learning. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2016.

[IM98a] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.

[IM98b] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the ACM symposium on Theory of computing*, 1998.

[IPRR16] Mohsen Imani, Daniel Peroni, Abbas Rahimi, and Tajana Rosing. Resistive CAM acceleration for tunable approximate computing. *IEEE Transactions on Emerging Topics in Computing*, 2016.

[JEB18] Amirhossein Jabalameli, Nabil Ettehadi, and Aman Behal. Edge-based recognition of novel objects for robotic grasping. *arXiv preprint arXiv:1802.08753*, 2018.

[JGCM+15] Cedric Jayet-Griffon, M-A Cornelie, Paolo Maistri, PH Elbaz-Vincent, and Régis Leveugle. Polynomial multipliers for fully homomorphic encryption on FPGA. In *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE, 2015.

[JKLS18a] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *ACM CCS*, 2018.

[JKLS18b] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS*

2018: 25th Conference on Computer and Communications Security, pages 1209–1222, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[jLS18] Wen jie Lu and Jun Sakuma. Faster multiplication triplet generation from homomorphic encryption for practical privacy-preserving machine learning under a narrow bandwidth. Cryptology ePrint Archive, Report 2018/139, 2018.

[JM18] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. *USENIX Security*, 2018.

[KG19] Alhassan KHEDR and Glenn Gulak. Homomorphic processing unit (HPU) for accelerating secure computations under homomorphic encryption, May 21 2019. US Patent App. 10/298,385.

[KGV16] Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. SHIELD: Scalable homomorphic implementation of encrypted data-classifiers. *IEEE Transactions on Computers*, 65(9):2848–2858, Sep. 2016.

[KLA19] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[KM11] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning*. Springer, 2011.

[KMP+17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *CCS*. ACM, 2017.

[KMR12] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In *CCS*, 2012.

[Kom05] Peter Komarinski. *Automated fingerprint identification systems (AFIS)*. Elsevier, 2005.

[KS08a] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, 2008.

[KS08b] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, 2008.

[KSK+18] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics*, 11(4):83, Oct 2018.

[KSMB13] Benjamin Kreuter, Abhi Shelat, Benjamin Mood, and Kevin RB Butler. PCF: A portable circuit format for scalable two-party secure computation. In *USENIX Security*, 2013.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security*, 2012.

[KSS14] Florian Kerschbaum, Thomas Schneider, and Axel Schröpfer. Automatic protocol selection in secure two-party computations. In *ACNS*, 2014.

[KTFY03] Gen Kasai, Yukihiro Takarabe, Koji Furumi, and Masato Yoneda. 200MHz/200MSPS 3.2 W at 1.5 V vdd, 9.4 Mbits ternary CAM with new charge injection match detect circuits and bank selection scheme. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2003.

[KW14] Toomas Krips and Jan Willemson. Hybrid model of fixed and floating point numbers in secure multiparty computations. In *ISC*, 2014.

[LDD+16] Ximeng Liu, Robert H Deng, Wenxiu Ding, Rongxing Lu, and Baodong Qin. Privacy-preserving outsourced calculation on floating point numbers. In *IEEE TIFS*, 2016.

[LHL+15] Maya Larson, Chunqiang Hu, Ruinian Li, Wei Li, and Xiuzhen Cheng. Secure auctions without an auctioneer via verifiable secret sharing. In *Proceedings of the 2015 Workshop on Privacy-Aware Mobile Computing*, pages 1–6. ACM, 2015.

[Lin16] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[liv] Indian Liver Patient Records, accessed on 01/20/2019. `https://www.kaggle.com/uciml/indian-liver-patient-records`.

[LJLA17a] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *CCS*, 2017.

[LJLA17b] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *ACM CCS*, 2017.

[LMIC14] Jing Li, Robert K Montoye, Masatoshi Ishii, and Leland Chang. 1 mb 0.41 μm^2 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing. *IEEE Journal of Solid-State Circuits*, 2014.

[LN16] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16: 15th International Conference on Cryptology and Network Security*, volume 10052 of *Lecture Notes in Computer Science*, pages 124–139, Milan, Italy, November 14–16, 2016. Springer, Heidelberg, Germany.

[LP00] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *CRYPTO*, 2000.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[LP12] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology*, 25(4):680–722, 2012.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[LSK+17] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[LTH+17] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th Annual ACM Symposium on Theory of Computing*, pages 1219–1234, New York, NY, USA, May 19–22, 2012. ACM Press.

[LWN+15] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. ObliVM: A programming framework for secure computation. In *IEEE S&P*, 2015.

[LZP17] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pages 345–353, 2017.

[mal] Malaria Cell Images, accessed on 01/20/2019. `https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria`.

[MF93] Anthony J McAuley and Paul Francis. Fast routing table lookup using CAMs. In *Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE*, 1993.

[MGC+16] Benjamin Mood, Debayan Gupta, Henry Carter, Kevin Butler, and Patrick Traynor. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *IEEE EuroS&P*, 2016.

[MHM+09] Shoun Matsunaga, Kimiyuki Hiyama, Atsushi Matsumoto, Shoji Ikeda, Haruhiro Hasegawa, Katsuya Miura, Jun Hayakawa, Tetsuo Endoh, Hideo Ohno, and Takahiro Hanyu. Standby-power-free compact ternary content-addressable memory cell chip using magnetic tunnel junction devices. *Applied Physics Express*, 2009.

[MKN+11] Shoun Matsunaga, Akira Katsumata, Masanori Natsui, Shunsuke Fukami, Tetsuo Endoh, Hideo Ohno, and Takahiro Hanyu. Fully parallel 6T-2MTJ nonvolatile TCAM with single-transistor-based self match-line discharge control. In *Symposium on VLSI Circuits (VLSIC)*, 2011.

[MMJP09] Davide Maltoni, Dario Maio, Anil K Jain, and Salil Prabhakar. *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009.

[MML+19] Mohsen Minaei, Mainack Mondal, Patrick Loiseau, Krishna Gummadi, and Aniket Kate. Lethe: Conceal content deletion from persistent observers. *Proceedings on Privacy Enhancing Technologies*, 2019(1):206–226, 2019.

[MMSK18] Mohsen Minaei, Pedro Moreno-Sanchez, and Aniket Kate. R3C3: Cryptographically secure censorship resistant rendezvous using cryptocurrencies. *IACR Cryptology ePrint Archive*, 2018:454, 2018.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay-secure two-party computation system. In *USENIX Security*, 2004.

[MOHO14] Ciara Moore, Máire O'Neill, Neil Hanley, and Elizabeth O'Sullivan. Accelerating integer-based fully homomorphic encryption using Comba multiplication. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6. IEEE, 2014.

[MOR16] Payman Mohassel, Ostap Orobets, and Ben Riva. Efficient server-aided 2PC for mobile phones. In *PoPETs*, 2016.

[MR18] Payman Mohassel and Peter Rindal. ABY3: a mixed protocol framework for machine learning. In *ACM CCS*, 2018.

[MSR+17] Vincent Migliore, Cédric Seguin, Maria Méndez Real, Vianney Lapotre, Arnaud Tisserand, Caroline Fontaine, Guy Gogniat, and Russell Tessier. A high-speed accelerator for homomorphic encryption using the karatsuba algorithm. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):138, 2017.

[MTK+16] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[MZ17a] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE S&P*, 2017.

[MZ17b] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE S&P*, 2017.

[nis17] National institute of standards and technology. `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf`, 2017.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, 1999.

[nuFer] nuFHE. `https://github.com/nucypher/nufhe`, NuCypher.

[NWI+13] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE S&P*, 2013.

[ÖDSS15] Erdinç Öztürk, Yarkin Doröz, Berk Sunar, and Erkay Savas. Accelerating somewhat homomorphic evaluation using FPGAs. *IACR Cryptology ePrint Archive*, 2015:294, 2015.

[ÖDSS16] Erdinç Öztürk, Yarkın Doröz, Erkay Savaş, and Berk Sunar. A custom accelerator for homomorphic encryption applications. *IEEE Transactions on Computers*, 66(1):3–16, 2016.

[OKK16] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[OPB07] Claudio Orlandi, Alessandro Piva, and Mauro Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security*, 2007(1):037343, 2007.

[PA01] Midas Peng and Sherri Azgomi. Content-addressable memory (CAM) and its network applications. In *International IC-Taipei Conference Proceedings*, 2001.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[PKUM16] Erman Pattuk, Murat Kantarcioglu, Huseyin Ulusoy, and Bradley Malin. CheapSMC: A framework to minimize secure multiparty computation cost in the cloud. In *DBSec*, 2016.

[PKV+14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. Blind seer: A scalable private dbms. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014.

[PLWZ19] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019.

[PNPM15] Thomas Pöppelmann, Michael Naehrig, Andrew Putnam, and Adrian Macias. Accelerating homomorphic evaluation on reconfigurable hardware. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 143–163. Springer, 2015.

[PS15] Pille Pullonen and Sander Siim. Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations. In *FC*, 2015.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security Symposium*, pages 515–530, 2015.

[QA08] Yinian Qi and Mikhail J Atallah. Efficient privacy-preserving k-nearest neighbor search. In *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, pages 311–319. IEEE, 2008.

[Rab05] Michael O Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005.

[RBK+14] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[RCK20] M Sadegh Riazi, Seyed Mohammad Chavoshian, and Farinaz Koushanfar. SynFi: Automatic synthetic fingerprint generation. 2020.

[RCS+16] M Sadegh Riazi, Beidi Chen, Anshumali Shrivastava, Dan Wallach, and Farinaz Koushanfar. Sub-linear privacy-preserving search with untrusted server and semi-honest parties. *arXiv preprint arXiv:1612.01835*, 2016.

[RDGK16a] M Sadegh Riazi, Neeraj KR Dantu, LN Vinay Gattu, and Farinaz Koushanfar. GenMatch: Secure DNA compatibility testing. In *IEEE Hardware Oriented Security and Trust (HOST)*, 2016.

[RDGK16b] M Sadegh Riazi, Neeraj KR Dantu, LN Vinay Gattu, and Farinaz Koushanfar. GenMatch: Secure DNA compatibility testing. In *IEEE HOST*, 2016.

[RGC+15] Abbas Rahimi, Amirali Ghofrani, Kwang-Ting Cheng, Luca Benini, and Rajesh K Gupta. Approximate associative memristive memory for energy-efficient GPUs. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2015.

[Ria16] M Sadegh Riazi. Large-scale privacy-preserving matching and search. Master's thesis, 2016.

[RIKR17] Mohammad Samragh Razlighi, Mohsen Imani, Farinaz Koushanfar, and Tajana Rosing. Looknn: Neural network with no multiplication. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017.

[Rin] Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. `https://github.com/osu-crypto/libOTe`.

[RJHK19] M Sadegh Riazi, Mojan Javaheripi, Siam U Hussain, and Farinaz Koushanfar. MPCircuits: Optimized circuit generation for secure multi-party computation. *IEEE Hardware Oriented Security and Trust (HOST)*, 2019.

[RJV+15] Sujoy Sinha Roy, Kimmo Järvinen, Frederik Vercauteren, Vassil Dimitrov, and Ingrid Verbauwhede. Modular hardware architecture for somewhat homomorphic function evaluation. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 164–184. Springer, 2015.

[RJV+18] Sujoy Sinha Roy, Kimmo Järvinen, Jo Vliegen, Frederik Vercauteren, and Ingrid Verbauwhede. HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation. *IEEE Transactions on Computers*, 67(11):1637–1650, 2018.

[RK18] M Sadegh Riazi and Farinaz Koushanfar. Privacy-preserving deep learning and inference. In *Proceedings of the International Conference on Computer-Aided Design*, page 18. ACM, 2018.

[RLPD20] M Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. HEAX: An architecture for computing on encrypted data. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[RMTR18] Aditi Roy, Nasir Memon, Julian Togelius, and Arun Ross. Evolutionary methods for generating synthetic masterprint templates: Dictionary attack in fingerprint recognition. In *2018 International Conference on Biometrics (ICB)*, pages 39–46. IEEE, 2018.

[ROC+18] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M Dai, Nissan Hajaj, Michaela Hardt, Peter J Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, Patrik Sundberg, Hector Yee, Kun Zhang, Yi Zhang, Gerardo Flores, Gavin E. Duggan, Jamie Irvine, Quoc

Le, Kurt Litsch, Alexander Mossin, Justin Tansuwan, De Wang, James Wexler, Jimbo Wilson, Dana Ludwig, Samuel L. Volchenboum, Katherine Chou, Michael Pearson, Srinivasan Madabushi, Nigam H. Shah, Atul J. Butte, Michael D. Howell, Claire Cui, Greg S. Corrado, and Jeffrey Dean. Scalable and accurate deep learning with electronic health records. *npj Digital Medicine*, 1(1):18, 2018.

[RORF16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[RRK18] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, page 2. ACM, 2018.

[RRK19] M Sadegh Riazi, Bita Darvish Rouhani, and Farinaz Koushanfar. Deep learning on private data. *IEEE Security and Privacy (S&P) Magazine.*, 2019.

[RSC+19] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E Lauter, and Farinaz Koushanfar. XONN: XNOR-based oblivious deep neural network inference. *USENIX Security*, 2019, 2019.

[RSK17a] M Sadegh Riazi, Mohammad Samragh, and Farinaz Koushanfar. CAMsure: Secure content-addressable memory for approximate search. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–20, 2017.

[RSK17b] M Sadegh Riazi, Ebrahim M Songhori, and Farinaz Koushanfar. PriSearch: Efficient search on private data. In *Design Automation Conference (DAC)*, 2017.

[RSS+17] M Sadegh Riazi, Ebrahim M Songhori, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Toward practical secure stable matching. *Proceedings on Privacy Enhancing Technologies*, 2017(1):62–78, 2017.

[RTJ+19] Sujoy Sinha Roy, Furkan Turan, Kimmo Jarvinen, Frederik Vercauteren, and Ingrid Verbauwhede. FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 387–398. IEEE, 2019.

[RWT+18a] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *ASIACCS'18*, 2018.

[RWT+18b] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 707–721. ACM, 2018.

[SCA17] SCAPI. the Secure Computation API. https://github.com/cryptobiu/libscapi, 2017.

[SEA] Microsoft SEAL (release 2.3). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA.

[SEA19] Microsoft SEAL (release 3.3). https://github.com/Microsoft/SEAL, June 2019. Microsoft Research, Redmond, WA.

[SHS+15] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. TinyGarble: Highly compressed and scalable sequential garbled circuits. In *IEEE S&P*, 2015.

[SHSK15] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. Compacting privacy-preserving k-nearest neighbor search using logic synthesis. In *Proceedings of the 52nd Annual Design Automation Conference*, page 36. ACM, 2015.

[SK11] Subashini Subashini and Veeraruna Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1):1–11, 2011.

[SKGK18] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In *International Conference on Machine Learning*, pages 4497–4506, 2018.

[SKK09] Mark Shaneck, Yongdae Kim, and Vipin Kumar. Privacy preserving nearest neighbor search. In *Machine Learning in Cyber Trust*. Springer, 2009.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. CVPR, 2015.

[SRH+19] Ebrahim M Songhori, M Sadegh Riazi, Siam U Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. ARM2GC: Succinct garbled processor for secure computation. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[SS08] Ahmad-Reza Sadeghi and Thomas Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *International Conference on Information Security and Cryptology*, pages 336–353. Springer, 2008.

[SS15] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *CCS*, 2015.

[SSSS17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *S&P*. IEEE, 2017.

[SSW09] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient privacy-preserving face recognition. In *International Conference on Information Security and Cryptology*. Springer, 2009.

[SW11] R. Sedgewick and K. Wayne. *Algorithms*. Pearson Education, 2011.

[SZ13] Thomas Schneider and Michael Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *FC*, 2013.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TMAJ] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. *Technical Report HPL-2008-20*.

[TPKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. Privacy-preserving error resilient DNA searching through oblivious automata. In *CCS*. ACM, 2007.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *USENIX Security*, 2016.

[Ver11] Damien Vergnaud. Efficient and secure generalized pattern matching via fast fourier transform. In *International Conference on Cryptology in Africa*. Springer, 2011.

[Wak68] Abraham Waksman. A permutation network. *Journal of the ACM (JACM)*, 15(1):159–163, 1968.

[WCH14] Wei Wang, Zhilu Chen, and Xinming Huang. Accelerating leveled fully homomorphic encryption using GPU. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2800–2803. IEEE, 2014.

[WCKM09] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. Secure knn computation on encrypted databases. In *Proceedings of the ACM SIGMOD International Conference on Management of data*, 2009.

[WCS15] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound. In *ACM CCS'15*, pages 850–861. ACM, 2015.

[WGC18] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: Efficient and private neural network training, 2018.

[WH13] Wei Wang and Xinming Huang. FPGA implementation of a large-number multiplier for fully homomorphic encryption. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 2589–2592. IEEE, 2013.

[WHC+12] Wei Wang, Yin Hu, Lianmu Chen, Xinming Huang, and Berk Sunar. Accelerating fully homomorphic encryption using GPU. In *2012 IEEE conference on high performance extreme computing*, pages 1–5. IEEE, 2012.

[WHC+13] Wei Wang, Yin Hu, Lianmu Chen, Xinming Huang, and Berk Sunar. Exploring the feasibility of fully homomorphic encryption. *IEEE Transactions on Computers*, 64(3):698–706, 2013.

[WHEW13] Wei Wang, Xinming Huang, Niall Emmart, and Charles Weems. VLSI design of a large-number multiplier for fully homomorphic encryption. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(9):1879–1887, 2013.

[WYDL18] Xintao Wang, Ke Yu, Chao Dong, and Chen Change Loy. Recovering realistic texture in image super-resolution by deep spatial feature transform. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[WYW+18] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. ESRGAN: Enhanced super-resolution generative adversarial networks. In *The European Conference on Computer Vision Workshops (ECCVW)*, September 2018.

[yah17] Yahoo security notice. `https://help.yahoo.com/kb/account/SLN27925.html`, 2017.

[Yao82] Andrew C Yao. Protocols for secure computations. In *FOCS*, 1982.

[Yao86a] A. Yao. How to generate and exchange secrets. In *FOCS*. IEEE, 1986.

[Yao86b] Andrew Yao. How to generate and exchange secrets. In *FOCS*, 1986.

[YLX13] Bin Yao, Feifei Li, and Xiaokui Xiao. Secure nearest neighbor revisited. In *IEEE International Conference on Data Engineering (ICDE)*, 2013.

[YSK+13] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure pattern matching using somewhat homomorphic encryption. In *Proceedings of the 2013 ACM workshop on Cloud computing security workshop*. ACM, 2013.

[ZE15] Samee Zahur and David Evans. Obliv-C: A language for extensible data-oblivious computation. *IACR Cryptology ePrint Archive*, 2015:1153, 2015.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *EUROCRYPT*, 2015.

[ZSB13] Yihua Zhang, Aaron Steele, and Marina Blanton. PICCO: a general-purpose compiler for private distributed computation. In *CCS*, 2013.