Hardware and Protocols for Authentication and Secure Computation

by

Chiraag Juvekar

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of Doctor of Philosophy

at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

© Massachusetts Institute of Technology 2018. All rights reserved.

Signature redacted

Author ..............................
Department of Electrical Engineering and Computer Science
May 23, 2018

Signature redacted

Certified by ..............................
Anantha Chandrakasan
Vannevar Bush Prof. of Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Accepted by ..............................
Leslie A. Akisluldezinski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students
Hardware and Protocols for Authentication and Secure Computation

by

Chiraag Juvekar

Submitted to the Department of Electrical Engineering and Computer Science on May 23, 2018, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

The Internet of Things has resulted in an exponential rise in the number of embedded electronic devices. This thesis deals with ensuring the security of these embedded devices. In particular we focus our attention on two problems: first we look at how these devices can convince another of their identity i.e. authentication and second we look at how these devices and cloud servers can compute joint functions of their private inputs while revealing nothing but the computation results to the other i.e. secure computation.

We start with the problem of counterfeit detection through electronic tagging. Physical access to electronic tags can be leveraged to mount side-channel and fault injection attacks. We design a new tagging solution that leverages ferro-electric capacitor based non volatile memory to addresses these issues. Next we note that resource constraints imposed by embedded devices often preclude the use of public-key cryptography. We address this issue through the development of a lightweight (10k-Gate) Elliptic Curve accelerator for the K-163 curves, which allows us to build a secure wireless-charging system that can block power from counterfeit and potentially dangerous chargers. Next we build upon these insights to develop a new authentication protocol which combines the leakage resilience and public-key authentication properties of our previous tagging solutions. We implement this bilinear pairing based protocol on a RISCV processor and demonstrate its practicality in an embedded environment through reuse of existing hardware accelerated cryptography for the TLS protocol.

The final part of this thesis develops a framework for secure two-party computation. Our primary contribution is a judicious combination of homomorphic encryption and garbled circuits to substantially improve the performance of secure two-party computation. This allows us to present a practical solution to the problem of secure neural network inference, i.e. classifying your private data against a server's private model without either party sharing their data with the other. Our hybrid approach improves upon the state-of-art by 20-30× in classification latency. Our final contributions are two efficient 2PC protocols that implement secure matrix multiplication.
and vector-OLE primitives. For both these tasks we improve concrete computation and communication performance over the state-of-art by an order of magnitude.

Thesis Supervisor: Anantha Chandrakasan
Title: Vannevar Bush Prof. of Electrical Engineering and Computer Science
To my parents, Rashmi and Shashikant.
Acknowledgments

I came to MIT six years ago as a fresh graduate student with a hazy idea of working on digital circuits. It was my advisor, Prof. Anantha Chandrakasan, who first suggested that I look into designing hardware for cryptography, Fully Homomorphic Encryption (FHE) in particular. Even back then it was painfully evident to me that I did not know much about either. It was perhaps for the best then, that I did not know how much I did not know, for with that knowledge I might not have even taken the first step. Therefore it is no exaggeration to say that this thesis would not have been possible without help and advice of many individuals.

Firstly, I would like to thank Prof. Anantha Chandrakasan, my research advisor for his mentorship and providing me the opportunity and freedom to work on these challenging and interesting problems. His attention to detail and dedication to the group has made my experience at MIT truly memorable. His depth of knowledge and sheer efficiency is baffling and it has been and privilege to be his student.

I would also like to thank Prof. Vinod Vaikuntanathan and Prof. Ronald Rivest for being a part of my thesis committee. It has been a great experience collaborating closely with Prof. Vaikuntanathan. If you want to learn anything about lattices the shortest-vector certainly points to his office. I would like to thank Prof. Rivest for both his time and critical feedback has helped immensely in the quality of this thesis.

This thesis certainly would have been impossible without the backing of numerous funding agencies. In particular I would like to thank the Stata Family, Texas Instruments, Qualcomm Innovation Fellowship and Delta Electronics for their support. Many of the projects that I have worked have involved chip fabrication which is predicated on the external fabrication support made possible through the generosity of TSMC and Texas Instruments.

I have been extremely lucky to have been part of many collaborative projects over the past six years and as result I have the chance to interact with some amazing circuit designers and cryptographers. I am thankful to both Mehul Tikekar and Chao-Tsung Huang who first introduced me to the wonderful and complex world of video coding during my first 2 years as a Master’s student. I have learnt a lot about mixed signal design from both Hyung-Min Lee and Joyce Kwong through a very fruitful collaboration on the Authentication Tag project. There would be no wireless power transmitter to secure if it were not for the collaboration with Nachiket Desai (neither would he have had to fly from Portland to Boston after graduating to work on the demo). I would like to thank both Prof. Yael Kalai and Prof. Vaikuntanathan for sharing their deep insights about leakage resilient cryptography. I am thankful to both Prof. Kurt Rohloff and Yuriy Polyanksy at NJIT for access to the PALISADE library. Working with Utsav Banerjee, Andrew Wright and Prof. Arvind was my first experience at software-hardware co-design and it has been a very enriching endeavour that has helped me evolve my perspective on designing ASICs. Perhaps, most surprising to me was the collaboration on a frequency hopping transmitter with Phil Nadeau, Rabia Tugce Yazicigil and Daniel Richman which helped me scratch the surface of what it means to design an RFIC. I would like to thank Sam Fuller for
his words of encouragement and many fruitful discussions on the state of security in the Internet of Things (IoT).

I am also grateful to Texas Instruments for granting me the opportunity to work as a summer intern. In particular I would like to thank Dennis Buss for setting up the internship, my supervisor Srinath Ramaswamy and my mentor Joyce Kwong for the amazing summer I spent there. I had was fortunate enough to work with some incredibly talented undergraduate students through summer internships and the SuperUROP program. Working with Shubham, Skanda, Lisa, Jay, Andrew and Leo was both great fun and immensely rewarding. I had the chance to TA 6.888 this year and I am thankful to Prof. Vivienne Sze and Prof. Joel Emer for teaching me so much about architecture and neural networks.

Discussions with various members of ananthagroup including Arun, Mehul, Nachiket, Phil, Priyanka and Ujwal have helped me better understand a wide variety of devices, circuits and algorithms concepts. Thanks for patiently answering all my silly questions. Any mention of the group would be remiss without thanking Margaret, our lab admin, who single handedly makes dealing with reimbursements and scheduling painless.

Beyond work I have made some amazing friends at MIT who have added balance and mirth to my MIT experience. Hanging out with Arun, Atulya, Ashwin, Divya, Dragos, Mukund, Mehul, Nachiket, Phil, Priyanka, Sanket, Sameer, Suhrich, Suvinay, Ujwal, Vaibhav, Viral and Vishnu has been great fun and I hope we continue to remain in touch.

Finally, I would like to my family, my wife Sai and my parents, Rashmi and Shashikant. If not for their constant support, motivation and countless sacrifices, I would not have made it thus far.
# Contents

1 Introduction ........................................... 19
   1.1 Motivation and Scope .............................. 19
      1.1.1 Authentication and Counterfeiting .......... 20
      1.1.2 Secure Two Party Computation and Data-Privacy 21
   1.2 Thesis Outline .................................. 21

2 Wireless Authentication Tags for Counterfeit Prevention 25
   2.1 Introduction ..................................... 25
   2.2 System Architecture .............................. 26
      2.2.1 Authentication System and Tag Overview .... 26
      2.2.2 Threat Model and Implemented Countermeasures 27
      2.2.3 Protocol Details ............................ 29
   2.3 Authentication Engine ........................... 31
      2.3.1 Overall Architecture ........................ 31
      2.3.2 Keccak Algorithm and Implementation ...... 32
      2.3.3 Selective NVDDF Storage ..................... 34
   2.4 Energy Backup and Power Glitch Countermeasures .... 35
      2.4.1 Area-Optimal Energy Backup Design ......... 35
      2.4.2 Energy Backup Unit .......................... 37
      2.4.3 Energy control during power glitches ...... 37
   2.5 Measurement Results and Conclusion .............. 39
      2.5.1 Chip Micrograph and Test Setup .............. 39
      2.5.2 Energy Backup Storage ....................... 39
3 Wireless Charger Authentication

3.1 Introduction ................................................. 45
3.2 Authentication Block ......................................... 47
  3.2.1 Authentication Protocol .................................. 48
  3.2.2 Elliptic Curve Scalar Multiplication (ECSM) Architecture Selection ................................. 49
  3.2.3 ECSM Implementation and Performance ......................... 51
3.3 Measurement Results ........................................ 53
  3.3.1 Test Setup .............................................. 53
  3.3.2 Detuning Mechanism ..................................... 54
  3.3.3 Charger Authentication ................................... 55
  3.3.4 Dynamic Power Balancing ................................ 55
3.4 Conclusion ..................................................... 58

4 Practical Leakage Resilient Identification Schemes ............... 61

4.1 Introduction .................................................. 61
  4.1.1 Threat Model ........................................... 63
4.2 Preliminaries .................................................. 64
  4.2.1 Leakage Assumptions ..................................... 64
  4.2.2 Bilinear Groups and Computational Assumptions .......... 64
  4.2.3 Barreto-Naehrig curves .................................. 65
  4.2.4 Identification Scheme ..................................... 66
  4.2.5 Schnorr ID ............................................... 67
4.3 Protocol ......................................................... 67
  4.3.1 Continual Leakage Resilient ID ............................ 68
  4.3.2 Security Guarantees, Computation Requirements and Practical Issues ............................... 70
4.4 Hardware Accelerated Implementation ............................ 72
5.7.3 The Hoisting Optimization (Halevi and Shoup) .................. 110
5.7.4 Additional Implementation Details .............................. 110
5.8 Fast Homomorphic Convolutions .................................. 111
5.8.1 More Techniques for Homomorphic Convolution ................. 116
5.9 Implementation and Micro-benchmarks ............................ 117
5.9.1 Cryptographic Primitives ....................................... 118
5.9.2 Evaluation Setup ............................................... 118
5.9.3 Micro-benchmarks ............................................... 119
5.9.4 Tuning the Relinearization Window ............................ 122
5.10 Network Benchmarks and Comparison ............................. 123
5.11 Conclusions and Future Work .................................... 126

6 Homomorphic Protocols for faster Matrix Multiplication and Vector-OLE

6.1 Matrix Multiplication ............................................. 127
6.1.1 Baseline Approach ............................................. 129
6.1.2 Using Gazelle ................................................ 129
6.1.3 Outer-Product Matrix Multiplication .......................... 130
6.1.4 Implementation ............................................... 130
6.2 Vector-OLE ...................................................... 131
6.2.1 Baseline Approach ............................................. 133
6.2.2 Optimizing Bandwidth ........................................ 134
6.2.3 Parameter selection for bigint-free rounding ................. 135
6.2.4 Rounding based function privacy ............................. 136
6.2.5 NTT optimization ............................................. 138
6.2.6 Implementation ............................................... 139

7 Conclusion and Future Work ........................................ 141
7.1 Summary and Contributions ....................................... 141
7.2 Future Work ...................................................... 144
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Authentication system consisting of the tag, the handheld reader and the back-end server.</td>
<td>27</td>
</tr>
<tr>
<td>2-2</td>
<td>Overview of the key features implemented on the authentication tag.</td>
<td>27</td>
</tr>
<tr>
<td>2-3</td>
<td>Block diagram showing our update protocol. $K_i$ are the per-query keys. $C_i$, $AD$, $R_i$ and $A_i$ are the challenge, associated data, response, and authentication message, respectively.</td>
<td>30</td>
</tr>
<tr>
<td>2-4</td>
<td>Overall architecture of the Authentication Engine (AE) circuits.</td>
<td>32</td>
</tr>
<tr>
<td>2-5</td>
<td>Modes of the Keccak algorithm implemented on the AE ($r$, $c$, and $f$ represent the rate, capacity, and permutation function, respectively)</td>
<td>33</td>
</tr>
<tr>
<td>2-6</td>
<td>Keccak architecture sharing a common combinational round implementation across the cryptographically secure pseudo random number generation (CS-PRNG) and authenticated encryption with associated data (AEAD) state.</td>
<td>34</td>
</tr>
<tr>
<td>2-7</td>
<td>Area-optimal energy backup storage design using the backup capacitor ($C_{BK}$) and the energy backup unit (EBU).</td>
<td>36</td>
</tr>
<tr>
<td>2-8</td>
<td>Block diagram of the EBU.</td>
<td>38</td>
</tr>
<tr>
<td>2-9</td>
<td>Power-glitch response as a function of the tag state.</td>
<td>39</td>
</tr>
<tr>
<td>2-10</td>
<td>Experimental Setup.</td>
<td>40</td>
</tr>
<tr>
<td>2-11</td>
<td>Measured waveforms showing safe shutdown with the EBU during a worst case power interruption event.</td>
<td>40</td>
</tr>
<tr>
<td>2-12</td>
<td>Measured waveforms showing safe system response to two successive power glitch events.</td>
<td>41</td>
</tr>
</tbody>
</table>
3-1 System architecture of a Wireless Power Transfer (WPT) system with multiple receivers connected to a single authenticated charger .......... 47
3-2 Overview of the Authentication Engine ............................... 48
3-3 Comparison of K-163 ECSM Architectures ........................... 50
3-4 Performance summary of the ECSM unit .............................. 52
3-5 Die micrograph and performance summary of implemented receiver . 54
3-6 Measurement setup for charger authentication and power balancing . 55
3-7 Authentication flowchart and transmitter and receiver traces during authentication .................................................. 56
3-8 Co-operative power sharing by receivers at different distances from the charger ....................................................... 57
3-9 Range of power ratios obtainable by co-operative action by the receivers 58

4-1 Overview of the RISCV processor ...................................... 73
4-2 Chip Specifications .......................................................... 74
4-3 Block diagram of hardware implementation of prime field modular arithmetic and elliptic curve cryptography .......................... 75
4-4 Test Setup ................................................................. 78
4-5 Code Size Breakdown ...................................................... 79

5-1 A Convolutional Neural Networks (CNN) with two Conv layers and one FC layer. Rectified Linear Unit (ReLU) is used as the activation function and a MaxPooling layer is added after the first Conv layer. 88
5-2 Single-input Single-output (SISO) convolutions and multi-channel Conv layers .................................................. 89
5-3 Ciphertext Structure and Operations. Here, $n$ is the number of slots, $q$ is the size of ciphertext space (so a ciphertext required $\lceil \log_2 q \rceil$ bits to represent), $p$ is the size of the plaintext space (so a plaintext can have at most $\lceil \log_2 p \rceil$ bits), and $\eta$ is the amount of noise in the ciphertext. 96
5-4 A Plaintext Permutation in action. The permutation \( \pi \) in this example swaps the first and the second slots, and also the third and fourth slots. The operation incurs a noise growth from \( \eta \) to \( \eta' \approx \eta + \eta_{\text{rot}} \). Here, \( \eta_{\text{rot}} \approx n \log q \cdot \eta_0 \) where \( \eta_0 \) is some small "base noise".

5-5 The Gazelle protocol consisting of alternating Packed Additively Homomorphic Encryption (PAHE) and Garbled Circuits (GC) layers.

5-6 Our combined circuit for steps (a), (b) and (c) for the non-linear layers. The "+" gates refer to an integer addition circuit, "-" refers to an integer subtraction circuit and the ">" refers to the circuit refers to a greater than comparison. Note that the borrow of the subtraction gates is used as the select for the first and last multiplexer.

5-7 The naïve method is illustrated on the left and the diagonal method of Halevi and Shoup [75] is illustrated on the right. The entries in a single color live in the same ciphertext. The key feature of the diagonal method is that no two elements of the matrix that influence the same output element appear with the same color.

5-8 Four example extended diagonals after accounting for the rotation group structure.

5-9 Padded SISO Convolution.

5-10 Packed SISO Convolution. (Zeros in the punctured plaintext shown in white.)

5-11 Diagonal Grouping for Intermediate Ciphertexts \( (c_i = c_o = 8 \text{ and } c_n = 4) \)

5-12 Decomposing a strided convolutions into simple convolutions \( (f_w = f_h = 3 \text{ and } s_x = s_y = 2) \)
List of Tables

2.1 State Dependent non-volatile flip-flop (NVDFF) Energy Consumption 29
2.2 Keccak core benchmarking for 400 and 1600 bit state sizes ............ 34
2.3 Impact of Keccak state size and selective NVDFF instantiation on AE
   area and backup energy (all area in GE) ......................... 35
2.4 Authentication Tag Specification .................................. 41
2.5 Effect of temperature on tag specifications ........................ 42
3.1 Comparison of low-resource ECSM architectures ........................ 51
3.2 Comparison of recent low-resource ECSM implementations .......... 53
4.1 Operation Counts and Communication Cost ............................ 71
4.2 $\mathbb{F}_p$ operation counts in pairing computation ............... 76
4.3 $\mathbb{F}_p$ operation counts in elliptic curve point addition in affine coordinates 77
4.4 Measurement results at 0.8 V and 16 MHz ............................. 79
5.1 Prime Selection for PAHE ............................................. 99
5.2 Comparison of matrix-vector product algorithms ....................... 104
5.3 Comparing SISO 2D-convolutions .................................... 113
5.4 Comparing multi-channel 2D-convolutions ............................. 116
5.5 Fast Reduction for Number Theoretic Transform (NTT) and Inv. NTT 119
5.6 FHE Microbenchmarks .................................................. 119
5.7 Permutation Microbenchmarks ......................................... 120
5.8 Matrix Multiplication Microbenchmarks ............................... 121
5.9 Hybrid Matrix Multiplication Window Sizing .......................... 122
Chapter 1

Introduction

1.1 Motivation and Scope

The work described in this thesis is the direct result of convergence of two overarching trends. First, the Internet of Things (IoT) promises to revolutionize our lives by allowing us to better sense and control the world around us. It is projected that by 2020 over 30 billion interconnected embedded devices will be deployed to help us accomplish this vision [116]. A majority of these devices are extremely resource constrained and need to operate on strict energy budgets. This growth in the number of connected edge devices brings us to the second trend. Ever-increasing amounts of private user data are being uploaded to remote cloud computing environments by these “smart” devices in order to offload processing.

The resource constraints inherent to embedded edge-devices and the need to preserve user privacy in the face of cloud computing lead us to ask a very natural question: “What security guarantees can be assured in an era of distributed sensing and computing?".

Security, broadly defined as the ability to guarantee a desired property in the presence of adversaries, comes in many forms. One may seek to keep ones data private (confidentiality), ones identity private (anonymity) or prevent/detect the tampering of ones data (integrity). Much of this thesis focuses on two specific problems: authentication i.e being able to convince another of ones identity and secure two-party
computation i.e. allowing two parties to compute a common function of their private inputs while revealing nothing but the function output to the other. One can view these two problems as complementary issues. The former tackles the issue of how can a cloud server trust that an edge node is indeed who it claims to be, while the latter addresses the edge devices’ rightful concern of the privacy of its data that it entrusts to the cloud for the purpose of computation.

1.1.1 Authentication and Counterfeiting

Authentication is a classic problem in cryptography and numerous full-fledged protocols like Kerberos [143], TLS [45], etc. have been developed to address this issue. Our interest in authentication is mainly derived from the point of view of counterfeit prevention. Counterfeiting is a major problem plaguing both the retail sector as well as global supply chains. In the past few years, multi-million dollar losses have been attributed to counterfeit automotive parts, aircraft parts, and pharmaceutical drugs. One solution to this problem is affixing these products with cryptography-enabled authentication tags [152]. However such an approach merely shifts the underlying problem to the prevention of counterfeiting of the electronic tags [108, 134].

Attackers typically posses physical access to these tags which greatly expands the available attack surface and thus necessitates a more careful analysis. The simplest attack one can envision is detaching a valid authentication tag from the original product and attaching it to a counterfeit. Such a strategy allows an attacker to sell a single counterfeit product per stolen tag. More worrying are attacks which allow an attacker to clone the tags en-masse. Physical access can often be leveraged to mount clever side-channel [90] and fault injection attacks [72] which accomplish just this. In the first part of this thesis we take a holistic view of this issue and propose a combination of protocol-design and circuit-design techniques that are applicable to resource constrained edge-devices.
1.1.2 Secure Two Party Computation and Data-Privacy

Fueled by the massive influx of data, sophisticated algorithms and extensive computational resources, modern machine learning has found surprising applications in such diverse domains as medical diagnosis [56,149], facial recognition [139] and credit risk assessment [8]. Often many of these tasks deal with sensitive data which the data owners would like to keep private.

Consider for instance the example of medical diagnosis. A large hospital with a wealth of data on, say, retinal images of patients can use techniques from recent works, e.g., [149], to train a learning algorithm that takes a retinal image as input and predicts the occurrence of a medical condition called diabetic retinopathy. Since the generated model encodes private patient information the hospital may be concerned about making the model public to maintain HIPAA compliance. Simply building a web service, that hosts the model and provides predictions, may fail to satisfy users of such a service who will be rightfully concerned about the privacy of the inputs they are providing to this web service.

Modern cryptography provides us with many tools, in particular fully homomorphic encryption [66] and interactive protocols like garbled circuits [158], that can help us address this issue of secure two party computation. Unfortunately the inherent overhead in these techniques makes it challenging to scale their use to real-world problems. This is the issue that we tackle in the latter part of this thesis.

1.2 Thesis Outline

As a whole the thesis provides solutions for low-cost hardware authentication and provides new efficient protocols for computing on encrypted data.

Radio Frequency Identification (RFID) based authentication systems are often cost-constrained and susceptible to physical attack. In Chapter 2 we present the design of a secure authentication tag with wireless power and data delivery optimized for compact size and near-field applications. Power-glitch attacks are mitigated through state backup on ferroelectric capacitor (FeCap) based non-volatile flip-flops.
The tag uses Keccak [23] (cryptographic core of SHA3) to update key material before each protocol invocation, limiting side-channel leakage to a single trace per key.

In Chapter 3 we present a resonant wireless charging receiver with an active detuning mechanism for controlling the received power, without any passive components being switched in or out. This detuning mechanism is then combined with a 10k NAND gate on-chip elliptic curve accelerator and in-band telemetry for authenticating a wireless charger using Elliptic Curve Cryptography (ECC), with up to 16× rejection at the output of the receiver. The accelerator itself is energy-efficient and achieves 0.77-μJ/elliptic curve scalar multiplication.

In Chapter 4, we improve on these authentication mechanisms by combining some of their best features and develop a new leakage resilient zero-knowledge identification (ID) scheme. Our protocol relies on bilinear pairings on Barreto Naehrig (BN) Elliptic Curves and we demonstrate the practicality of this protocol in an embedded environment by implementing it on a low-end RISCV [154] processor. We reuse traditional cryptographic accelerators aimed for Transport Layer Security (TLS) and demonstrate real-time (<3s) prover run-time which can further be shortened to <100ms through pre-computation at sub-1mW power consumption.

In Chapter 5, we present Gazelle, a scalable and low-latency system for secure neural network inference, using an intricate combination of homomorphic encryption and traditional two-party computation techniques (such as garbled circuits). We evaluate our protocols on benchmark neural networks trained on the MNIST and CIFAR-10 datasets and show that Gazelle outperforms the best existing systems such as MiniONN [99] by 20 times and Chameleon [128] by 30 times in online runtime. Similarly when compared with fully homomorphic approaches like CryptoNets [68] we demonstrate three orders of magnitude faster online run-time.

We then build on top of Gazelle and demonstrate how to construct secure interactive protocols for two building block algorithms in Chapter 6. First we demonstrate a new outer-product based protocol for matrix-matrix multiplication that is (8-25×) faster than than the state-of-the art [100] and can be within a 3× overhead of plaintext
computation. Next we implement a new big-integer free vector-OLE protocol using a new rounding based function privacy mechanism. We show how to select primes compatible with our new rounding based scheme and reduce concrete communication-overhead to within $4\times$ of an insecure plaintext implementation. This improves the state-of-art [10] by $8\times$ on computation and $4\times$ on communication.

Finally we summarize our contribution and learnings in Chapter 7 and identify some avenues for future investigation.
Chapter 2

Wireless Authentication Tags for Counterfeit Prevention

2.1 Introduction

Electronic tags affixed to components can offer a solution for authenticating them at the point of purchase [148]. Unfortunately, this also has the side effect of making the tags themselves a lucrative target for counterfeiters. Due to the embedded nature of these tagging solutions, protecting them necessitates effective countermeasures against physical attacks [151] such as fault-injection [72] and side-channel attacks [90]. This makes the design of a secure authentication tag a challenging and interesting problem.

Recent work on the design of RFID-like authentication tags can be roughly classified into two categories. First we have simple low-cost organic RFID tags that can be integrated on flexible substrates [113] [59]. These tags typically do not integrate any security features or programmable non-volatile memory due to technology limitations [112]. Second we have CMOS tags that are designed for more complex use-cases such as those where security is a concern [88] [55]. Here, cryptographic primitives are implemented to support authenticated modes of operation. Even when secure algorithms are used, the tags may still be vulnerable to implementation based attacks since the attacker can leverage physical access to the tag [117]. The cost-sensitive na-
tecture of RFID systems makes deploying circuit-level countermeasures against physical attacks challenging. Algorithmic countermeasures [105] [122] that use key update to provide physical attack resilience are a viable solution to this problem.

Our goal for this project is to build on these ideas and leverage circuit and technology innovations to enable one of the first physical embodiments of a wireless tagging solution using a low overhead key update protocol. Our key contributions are summarized below:

1. A cryptographic challenge-response protocol is implemented to allow a server to securely authenticate the tag.

2. The tag uses Keccak [23], the cryptographic core of SHA3, to update cryptographic keys before each protocol invocation, to prevent side-channel attacks.

3. Power-glitch attacks are mitigated via state backup on FeCap based NVDFFs [124] using the on-chip energy backup storage.

2.2 System Architecture

2.2.1 Authentication System and Tag Overview

Figure 2-1 shows the three major components of our authentication system: authentication tags which are affixed to the equipment being protected, a reader to interrogate these tags and verify their authenticity, and a back-end server which seeds the tags with an initial secret during an enrollment step at manufacture and then maintains a database in order to validate tag responses.

The authentication tag consists of an on-chip authentication engine that supports the required cryptographic protocols and a Wireless Power and Data Transfer (WPDT) frontend that harvests energy from the reader using a 433 MHz near-field inductive link. The authentication engine implements a per-query key update protocol using the Keccak algorithm to provide cryptographically secure pseudo random number generation (CS-PRNG) and authenticated encryption functionalities. NVDFFs
Figure 2-1: Authentication system consisting of the tag, the handheld reader and the back-end server.

are used to implement all the cryptographic state on the tag. A regulating voltage multiplier is implemented for efficient power conversion, and pulse-based modulation is used to reduce wireless power dead-time during telemetry. When combined with an on-chip energy backup unit, the NVDFFs provide robust countermeasures against power-glitch attacks. Figure 2-2 presents a block diagram of the tag summarizing these contributions.

Figure 2-2: Overview of the key features implemented on the authentication tag.

2.2.2 Threat Model and Implemented Countermeasures

The proposed tags are meant to protect relatively low-cost equipment. Hence the threat model that we consider in this work is limited to those attacks where the amortized cost of an attack is lower than the cost of the equipment being authenticated, such as:
1. Passive side-channel attacks: The attacker may passively monitor the power consumed and the EM emission radiated by the tag to mount a differential power analysis (DPA) [89] attack or a differential EM analysis (DEMA) [63] attack.

2. Active power glitch attacks: The attacker may introduce over-voltage or under-voltage power supply glitches to leak bits by inducing faults during cryptographic [72] or NV memory operation.

3. Protocol attacks on tag-reader communication: The attacker may capture, corrupt, and replay any transaction between the tag and the reader.

Physical probing and photon emission attacks can potentially be used for extracting secret key material. However, repeating such attacks on a per tag basis would be cost-prohibitive for the class of equipment we aim to protect with the proposed tags, and hence are not addressed here.

The NVDFFs used in this work exhibit state dependent power consumption during both regular operation and backup modes. As shown in Table 2.1, during the regular mode of operation the three sets of transitions \((0 \rightarrow 1), (1 \rightarrow 0)\) and \({(0 \rightarrow 0), (1 \rightarrow 1)}\) are clearly identifiable by their respective power consumption. Similarly backing up and restoring a ‘0’ and ‘1’ requires different amounts of energy. Hence, side-channel countermeasures are necessary. Simple power analysis (SPA) attempts to recover key material based on direct inspection of single traces. These attacks are most effective on small data-paths where the power trace information can be correlated to the Hamming weight of an intermediate result [101] or when control flow of the algorithm depends on intermediate values [57]. Our Authentication Engine (AE) implementation is constant time with no-state dependent control flow. In both regular as well backup modes of operation all 400 NVDFFs are updated in parallel on every cycle to prevent SPA attacks.

DPA-style attacks that involve the collection of multiple power traces with varying plaintexts for the same key [90] are still a potential attacker vector. These attacks work because specific points on the power trace are well correlated with a subset of key
Table 2.1: State Dependent NVDDFF Energy Consumption

<table>
<thead>
<tr>
<th>Initial State</th>
<th>Final State</th>
<th>Regular Mode Energy/bit (fJ)</th>
<th>Backup-Restore Energy/bit (fJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>7.660</td>
<td>2.501</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>52.454</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>30.344</td>
<td>1.402</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>7.365</td>
<td></td>
</tr>
</tbody>
</table>

bits. By trying various guesses for those key bits and correlating them with measured data, we observe that the correct guess correlates more strongly with measured power after certain threshold number of traces is reached. We protect against these attacks by deploying a protocol which ensures that the same key is never used twice for any cryptographic operation. Thus multiple power traces with the same key cannot be collected.

In practice the implementation of such a key update mechanism is complicated by the fact that the tags are passively powered by a reader which may be hostile. An attacker may interrupt the power supply during the key update operation hoping that the tag will either revert to using the old key or repeat the key update. If this happens, the tag performs multiple cryptographic computations using the same key making side-channel attacks possible. Similar attacks have been demonstrated in the context of secure EEPROM memories [15]. We mitigate these classes of attacks by using NVDDFFs to store both the cryptographic state as well as the relevant control registers on the tag. In case power is interrupted, all computation is simply paused, and the state is backed up to the non-volatile memory [124]. On the next power-up, the state is restored, and computation continues seamlessly without repeating even a single cycle of cryptographic computation.

2.2.3 Protocol Details

The server initializes each tag with a 64-bit public Chip ID and a unique 256-bit secret seed. Ideally, after this seed is written into the NVDDFFs, a single-bit fuse could be burned indicating that seeding is disabled. In order to reduce design complexity, the
corresponding fuse is not implemented in our test-chip. At a high-level, an iteration of the challenge-response protocol consists of the following steps. On receiving power from the reader, the tag first charges its backup energy storage before the AE is activated. Next the AE wakes up and updates its key using the Keccak-based CS-PRNG, increments the key index, and indicates to the reader that it is ready for the next challenge. The reader then sends the tag a random challenge to which the tag responds using an authenticated encryption with associated data (AEAD) mode of operation [24]. The reader then sends this challenge-response pair to a trusted server for validation. Figure 2-3 depicts our per-query key update protocol.

![Block diagram showing our update protocol.](image)

Given the initial seed values, it is possible to clone tags, and hence it is crucial that the server verification database is kept secret. The use of a CS-PRNG to generate fresh keys mitigates side channel and replay attacks. AEAD allows us to support both tag authentication as well as support future tag versions where the tag can transmit secret information, such as the state of on-chip sensors, back to the server. Both the CS-PRNG and AEAD modes use the Keccak algorithm [24], [25], and are
implemented by sharing common hardware on the tag. Both protocol and NVDDFF controllers are implemented as finite-state machines (FSMs) on chip. The AEAD authentication message is computed using the chip ID, key index, and challenge as associated data. Using the public chip ID and the current key index in the associated data also mitigates any de-synchronization attempts. Another advantage of sending the authenticated key index back to the server is that the server can verify that the key indices used in subsequent transactions form a monotonically increasing sequence. Thus, even if an attacker clones a tag, he must ensure that key indices across all the cloned copies are synchronized to prevent detection. In the absence of such synchronization, the server can easily detect that the sequence numbers in subsequent authentication attempts from the desynchronized clones that are not monotonically increasing and then invalidate their tag in the database.

On the other hand, the use of the chip ID and key index as associated data also implies that the protocol is no longer privacy preserving. A privacy-preserving protocol that supports key update on the tag without the need for server involvement in the key update is an interesting direction for future work. Finally, we note that the use of a 64-bit counter and a minimum time interval of 200 μs between 2 power-cycle events (imposed by the speed of the save-restore circuits) imply that counter-overflow based denial of service attacks are infeasible.

2.3 Authentication Engine

2.3.1 Overall Architecture

Figure 2-4 shows the overall architecture of the AE. The AE includes the Keccak cryptographic core consisting of the state for the CS-PRNG and AEAD modes as well as shared logic implementing the rounds of the Keccak algorithm. The state elements for the CS-PRNG mode, the chip ID, and the key-index are all implemented using a total of 571 NVDDFFs. The backup-restore and clock controllers interface these NVDDFFs with the WPDT block and implement power-glitch attack countermeasures.
A separate deprogramming and tamper detection block computes a checksum over all the NVDF data and erases all the state in case any mismatch is detected. The AE no longer contains any cryptographic material and must be re-seeded by the server before it can respond to any future challenges. The AE, including the NVDFs, occupies a total of 17.9k NAND gates. Operating at a 125 kHz clock frequency, it consumes 3.6 $\mu$W in standby and 8.6 $\mu$W when running the authentication protocol.

![Figure 2-4: Overall architecture of the AE circuits.](image)

### 2.3.2 Keccak Algorithm and Implementation

The Keccak algorithm implemented on the tag consists of two major components: an internal state and a permutation function that iteratively operates on that state. It supports two operations: absorb and squeeze and is commonly referred as a sponge construction [23]. The CS-PRNG and AEAD modes for our authentication protocol can be implemented by combining these operations as shown in Figure 2-5. Keccak state can be divided into rate and capacity components with the former determining throughput while the latter determining the security level. The permutation function consists of multiple rounds where each round includes five sub-rounds: $\theta$, $\rho$, $\pi$, $\chi$ and $\iota$.

We implement the Keccak-f[400] variant of the permutation function which uses a 400 bit internal state, compared to the 1600 bit state used in SHA3, in order to save area and backup energy. The rate and capacity sizes are set to 128 and 272 bits.
Figure 2-5: Modes of the Keccak algorithm implemented on the AE ($r$, $c$, and $f$ represent the rate, capacity, and permutation function, respectively) respectively (compared to 1024 and 576 in SHA3). Although the reduction in rate reduces the throughput we are still able to scan an average 30 tags/sec. Since the tag can only be seeded once at initialization we need to only consider the passive state-recovery attack analysis [24]. This results in 272-bit security up to $2^{64}$ iterations of the protocol. Thus the best attack on the CS-PRNG is to instead guess the initial 256-bit seed. Use of a 128-bit keys and authentication messages for AEAD results in 128-bit security against key and plaintext recovery as well as authentication forgery [25]. Thus even with reduced capacity 128-bit security is guaranteed.

The number of rounds used in the permutation function is set to 20 for both modes following the conservative guidelines used in SHA3 as opposed to the reduced round counts suggested in [25]. Since the internal state of the CS-PRNG must be persisted when the AEAD mode responds to the reader challenges, a separate 400-bit state-array is used for AEAD state. Each array is organized as 25 16-bit lane shift-registers allowing us to access the state one slice at a time over 16 cycles. Of the 5 sub-rounds used in the permutation, four ($\theta$, $\pi$, $\chi$ and $\iota$) act on the full slice and hence are they are regrouped and implemented in a common combinational block shared by both modes. The $\rho$ operation acts on lanes and is implemented with lane-specific hard-coded multiplexers. The Keccak architecture and the state array are presented in Figure 2-6. Since most literature cites performance metrics for only the core hashing functionality of Keccak, a comparison of our architecture when resynthesized to perform only hashing (and state implemented with regular flip-flops)
Table 2.2: Keccak core benchmarking for 400 and 1600 bit state sizes

<table>
<thead>
<tr>
<th>State</th>
<th>Reference</th>
<th>Tech. (nm)</th>
<th>Cycles</th>
<th>Gates (GE)</th>
<th>Power (µW/MHz)</th>
<th>Throughput (kb/s)</th>
<th>Energy /bit (pJ/bit)</th>
<th>Gate Efficiency (kbps/kGE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>This Work</td>
<td>130</td>
<td>356</td>
<td>2421</td>
<td>13</td>
<td>404.49</td>
<td>32.14</td>
<td>167.08</td>
</tr>
<tr>
<td></td>
<td>[48]</td>
<td>250</td>
<td>1000</td>
<td>3113</td>
<td>-</td>
<td>144</td>
<td>-</td>
<td>46.26</td>
</tr>
<tr>
<td></td>
<td>[87]</td>
<td>130</td>
<td>1000</td>
<td>5090</td>
<td>11.5</td>
<td>144</td>
<td>79.87</td>
<td>28.29</td>
</tr>
<tr>
<td>1600</td>
<td>This Work</td>
<td>130</td>
<td>1624</td>
<td>7537</td>
<td>41.2</td>
<td>827.59</td>
<td>49.78</td>
<td>109.8</td>
</tr>
<tr>
<td></td>
<td>[87]</td>
<td>130</td>
<td>1200</td>
<td>20790</td>
<td>44.9</td>
<td>1120</td>
<td>40.09</td>
<td>53.87</td>
</tr>
<tr>
<td></td>
<td>[120]</td>
<td>130</td>
<td>15427</td>
<td>5898</td>
<td>13.7</td>
<td>87.12</td>
<td>157.25</td>
<td>14.77</td>
</tr>
</tbody>
</table>

is presented in Table 2.2.

Figure 2-6: Keccak architecture sharing a common combinational round implementation across the CS-PRNG and AEAD state.

### 2.3.3 Selective NVDFF Storage

The proposed tag requires low-power NV memory for save-restore operation and large on-chip capacitors for energy backup storage. To address these design constraints, the tag was implemented in a 130 nm process which offers ferroelectric RAM and capacitor options. The AE uses a standard-cell based design, and the set of cells is augmented with an NVDFF cell. The NVDFF cell consumes $3.2 \times$ area of a conventional flip-flop and requires $3.4 \text{ pJ}$ of energy for the save-restore operation. This energy must be stored on-chip for security since it is used to mitigate power-glitch attacks. Thus, reducing the number of NVDFFs allows us to reduce the area occupied by both the AE as well the backup capacitors. Fortunately, only the chip ID, key index counter, the CS-PRNG mode state and associated control registers need to be saved to recover from a power glitch event. In particular, if a glitch occurs when a response is being generated, all the state associated with the current challenge can be safely erased. When power is restored the tag can update its key and simply respond to a new
Table 2.3: Impact of Keccak state size and selective NVDDF instantiation on AE area and backup energy (all area in GE)

challenge. Hence the AEAD state and other miscellaneous state elements are only implemented using regular flip-flops. Table 2.3 describes the impact of the reduction in the CS-PRNG state size (400 bits vs 1600 bits in SHA3) and selective NVDDF instantiation on the backup energy and AE area.

2.4 Energy Backup and Power Glitch Countermeasures

This section describes joint-work with Hyung-Min Lee who designed the WPDT subsystem of the authentication tag. The area-optimal capacitor sizing described below is the result of collaborative work while the energy backup unit (EBU) was contributed by Hyung-Min. A complete description of the WPDT system is available in [95].

2.4.1 Area-Optimal Energy Backup Design

In case wireless power is interrupted, the energy backup storage provides the energy required for both the NVDDF save-restore as well as key-update. The following three constraints need to be satisfied for safe-operation: total backup energy of 3.5 nJ must be supplied, VDD droop must be regulated within 10%, and finally the on-chip decoupling capacitor \( C_L \) must store 0.5 nJ to support the instantaneous current resulting from the parallel save-restore of all 571 NVDDFs. One approach is to size
$C_L$ to also meet the first two constraints, according to following equation.

$$E_{CL} = \frac{1}{2} \cdot C_{LVF} \cdot A_{CL} \cdot (V_{DD}^2 - V_{DD_{min}}^2)$$

Here $E_{CL}$ is the required energy (3.5 nJ), $C_{LVF}$ is the capacitance per unit area, $A_{CL}$ is the on-chip area of $C_L$, and $V_{DD_{min}}$ is set to 1.35 V (10% max. droop). Even when using high-density low-voltage ferroelectric capacitors (1.5 LV FeCaps), this approach consumes 0.8 mm$^2$ silicon area.

A more area efficient approach is to separate the decoupling (instantaneous energy) and backup (total energy) requirements. In this approach, we continue to use the LV FeCaps for decoupling but use high-voltage ferroelectric capacitors (3.3V HV FeCaps) for backup. The main advantage is that although the HV FeCaps have lower capacitance per unit area, $C_{HV,F}$, they allow higher energy densities since they can be charged to a higher voltage compared to $C_L$.

![Figure 2-7: Area-optimal energy backup storage design using the backup capacitor ($C_{BK}$) and the EBU](image)

Figure 2-7 describes the design methodology for area-optimal energy backup design. At startup the EBU charges $C_{BK}$ to 2.75 V. On interruption of wireless power, energy stored in $C_{BK}$ is supplied to the AE through a linear regulator. The EBU details are described in next paragraph $C_{BK}$ and $C_L$ are sized to satisfy the following
equations,

\[ E_{\text{total}} = E_{CL} - E_{EBU} + \eta_{\text{reg}} \cdot E_{CBK} \]

\[ = E_{CL} - E_{EBU} + \eta_{\text{reg}} \cdot \left( \frac{1}{2} \cdot C_{HV} \cdot A_{CBK} \cdot (V_{BK}^2 - V_{DDmin}^2) \right) \]

Here \( E_{\text{total}} \) is the required energy (3.5 nJ), \( E_{CL} \) is the decoupling energy (0.5 nJ), \( E_{EBU} \) is the energy consumption of the EBU, \( E_{CBK} \) is the available energy from \( CBK \), \( \eta_{\text{reg}} \) is the efficiency of the EBU regulator, and \( A_{CBK} \) is the on-chip area of \( CBK \). \( \eta_{\text{reg}} \) can be averaged to 79\% when \( CBK \) is discharged from \( V_{BK} \) (2.75 V) to \( V_{DDmin} \) (1.35 V). \( E_{EBU} \) was simulated to be 0.2 nJ, the EBU area, \( A_{EBU} \), was found to be 0.04 mm\(^2\) post layout, and \( CL \) is sized to 0.11 mm\(^2\) to supply 0.5 nJ. By solving the above equation, we find that \( CBK \) must be sized at 0.21 mm\(^2\). Thus, total area for energy backup including \( CL, CBK \) and the EBU is just 0.36 mm\(^2\), which is 2.2\times lower than the conventional approach of using a single high-density LV FeCap.

### 2.4.2 Energy Backup Unit

Figure 2-8 shows the block diagram of the EBU. The EBU has three operation modes: charging, standby, and backup. In the charging mode, the EBU uses wireless power to charge \( CBK \) to 2.75 V through a voltage doubler supplied by the Regulating Voltage Multiplier (RVM) at 1.5 V. In the standby mode, the EBU holds the charging voltage of \( CBK \) to 2.75 V by automatically refreshing through the same voltage doubler, while the AE is powered through the RVM. If the input power sensing circuit detects loss of wireless power, the EBU disables the voltage doubler and enters the backup mode. Then, \( CBK \) powers \( V_{DD} \) through a linear regulator until the AE completes key update and safe shutdown.

### 2.4.3 Energy control during power glitches

The RVM regulates power glitches overshoot to within 10\% of \( V_{DD} \). The AE was designed and experimentally verified to tolerate this level of \( V_{DD} \) overshoot. An under-
voltage glitch shorter than 0.5 μs is interpreted as a spurious bit by the telemetry circuit and ignored since it is not a part of a well-formed packet. The power glitch response to a longer under-voltage glitch depends on the exact state of the tag when power is lost as shown in Figure 2-9. Initially the AE is first held in reset while the on-chip energy backup capacitors are charged. After this the AE requests trickle charging of the internal $V_{DD,NV}$ rail in preparation for an NVDDF restore. If power is lost before this completes, no action needs to be taken, as the NVDDFs have not been accessed yet. Once $V_{DD,NV}$ reaches 1.5V, and NVDF restore is started, any power-loss at this point necessitates that the restore, a key-update cycle, and save all be performed from the energy stored on the backup capacitor. After the NVDDFs are restored, the AE tries to complete the key-update before running the challenge-response protocol. Power loss at this point is handled separately based on whether the AE was running in the CS-PRNG mode or AEAD mode. The former indicates that a key-update was in progress and must be resumed later, while the latter indicates that the previous challenge was aborted and a new key-update must be started. Finally, glitches during the save operation are ignored and the ongoing save is completed from the backup capacitor.
2.5 Measurement Results and Conclusion

2.5.1 Chip Micrograph and Test Setup

The authentication tag was fabricated in a 130 nm CMOS process and occupies 0.77 mm² area. Figure 2-10a shows the chip micrograph and floorplan including the WPDT circuits, EBU, AE, and on-chip capacitors, $C_L$ and $C_{BK}$. Figure 2-10b shows the test setup with the tag and the reader controlled by the backend server software. The tag is attached to an 8-mm Printed Circuit Board (PCB) coil ($L_2 = 35 \text{ nH}$) and wirelessly scanned by a discrete handheld reader with a 10-mm power coil ($L_1 = 23.5 \text{ nH}$). The reader is controlled by an Opal Kelly XEM6001 Field Programmable Gate Array (FPGA) board and transfers wireless power and data to scan the tag over a 433-MHz inductive link across a 5 mm separation. The backend server runs on a laptop connected to the reader to demonstrate tag enrolment and authentication.

2.5.2 Energy Backup Storage

Safe shutdown operation with the energy backup storage was verified through a worst-case power interruption event. Figure 2-11 shows the measured waveforms for safe shutdown when wireless power is interrupted just after $V_{DD, NV}$ reaches 1.5 V but before NVDFF restore starts. In response the WPDT circuits enter a sleep mode, and the energy from $C_{BK}$ is supplied to the AE to restore state, run a key-update step, and complete the save to the NVDFFs within 70 μs. The $V_{BK}$ decreases from 2.75 V as these operations are performed, but $V_{BK}$ does not drop below 1.5V indicating that the energy backup storage was sufficient to complete the safe shutdown. Then, the supply control unit initiates the VDDNV fast discharge for correct NVDFF operation.
(a) Chip micrograph.

(b) Test setup with the tag and reader (top) controlled by a back-end server software (bottom)

Figure 2-10: Experimental Setup.

Figure 2-11: Measured waveforms showing safe shutdown with the EBU during a worst case power interruption event
### Overall System vs. Authentication Engine

<table>
<thead>
<tr>
<th>Overall System</th>
<th>Authentication Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process</strong></td>
<td>130nm CMOS</td>
</tr>
<tr>
<td><strong>$V_{DD} / V_{BK} / P_{SB}$</strong></td>
<td>1.5V / 2.75 V / 7.5µW $^a$</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>30 Tags/sec</td>
</tr>
<tr>
<td><strong>Die Area</strong></td>
<td>0.77mm$^2$</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>Keccak (400 bit permutation)</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>433 MHz inductive link</td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td>1.79 kGE (incl. 571 NVDFFs)</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>3.6µW Standby / 8.6µW Auth</td>
</tr>
</tbody>
</table>

### Wireless Power & Data Telemetry

<table>
<thead>
<tr>
<th>Source</th>
<th>433 MHz inductive link</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_1 / L_2$</td>
<td>23.5nH / 35nH</td>
</tr>
<tr>
<td><strong>Downlink</strong></td>
<td>0.16 µs PPM-OOK, 125kb/s</td>
</tr>
<tr>
<td><strong>Uplink</strong></td>
<td>0.5 µs PW-LSK</td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td>1.79 kGE (incl. 571 NVDFFs)</td>
</tr>
<tr>
<td><strong>Power glitch</strong></td>
<td>NVDDF + Energy backup</td>
</tr>
</tbody>
</table>

$^a$ Static standby power  
$^b$ Including on-chip decoupling capacitors, $C_L$ and $C_{BK}$

**Table 2.4: Authentication Tag Specification**

#### 2.5.3 Power Glitch Response

Successful operation in the presence of two successive power-glitch events is shown in the measured waveforms of Figure 2-12. The first glitch is inserted during the AEAD mode, and the current response is aborted. The power is restored, and then a second glitch is inserted when the CS-PRNG is updating the key. The key-update is paused and completed in the next time the tag powers up. The tag then indicates that it is ready for a new challenge and successfully completes a challenge response iteration verifying that it can be recovered from the two glitches. Table 2.4 summarizes the specifications of the wireless authentication tag.

![Figure 2-12: Measured waveforms showing safe system response to two successive power glitch events.](image)

In simulation, the backup energy for a single NVDDF varies 10% over temperature from -20°C to 80 °C. Thus it is expected that available energy margin (+40%) on
<table>
<thead>
<tr>
<th>Temperature (°C)</th>
<th>$V_{DD}$ (V)</th>
<th>$V_{BK}$ (V)</th>
<th>Tag Clock Period (μs)</th>
<th>Backup-Energy* (pJ/Bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>1.465</td>
<td>2.660</td>
<td>10.13</td>
<td>2.30</td>
</tr>
<tr>
<td>20</td>
<td>1.496</td>
<td>2.716</td>
<td>8.38</td>
<td>2.48</td>
</tr>
<tr>
<td>40</td>
<td>1.508</td>
<td>2.732</td>
<td>7.72</td>
<td>2.52</td>
</tr>
<tr>
<td>80</td>
<td>1.530</td>
<td>2.754</td>
<td>7.02</td>
<td>2.58</td>
</tr>
</tbody>
</table>

*Simulated energy for backup-restore of 1 NVFF

Table 2.5: Effect of temperature on tag specifications

the backup capacitor is sufficient for correct operation. Full system operation was experimentally verified in a thermal chamber across this range of temperatures. Correct system glitch responses were verified, and measurement results of the VDD, VBK, and the on-chip ring-oscillator clock period are presented in Table 2.5. The clock recovery algorithm on the reader was adjusted to accept a wider range to account for the temperature drift of the oscillator.

2.5.4 Conclusion

In conclusion we developed a wireless authentication tag using FeCaps NVFFs for security applications and demonstrated mitigation techniques against passive and active threat models. The tag performs per-query key updates before each protocol invocation to prevent side-channel attacks. Also, the NVFF key storage and FeCaps-based energy backup solution enable complete NVFF save-restore for safe shutdown against power-glitch attacks. The tagging solution described in this does not depend on any off-chip energy-storage capacitors to provide these security guarantees. The proposed authentication tag can provide a secure proof of origin in a highly globalized supply chain.

We briefly describe some attack vectors not explicitly covered by these countermeasures. Brute-force attacks such as physically detaching the tag from the product or destroying the tag are not covered herein and would call-for innovative “analog-domain” countermeasures. Secondly our side-channel protection scheme does not cover template attacks [34] or algebraic side-channel attacks [125]. In these attacks, the attacker is able to arbitrarily program the tags with keys of his choosing in order
to characterize the power/EM profile. After this characterization phase is attacker is provided a single challenge trace from which he must recover the key. Our tags are not intended to be user programmable and hence do not implement countermeasures against these attacks. Both these attacks offer very interesting directions for future work.
Chapter 3

Wireless Charger Authentication

This section describes joint-work with Nachiket Desai who designed the wireless power delivery sub-system and a novel detuning mechanism. The contribution of this thesis is an ECC accelerator and the corresponding the authentication block which leverages this detuning mechanism to develop a secure wireless power receiver. The power balancing experiments in Section 3.3.4 are the result of collaborative effort. A complete description of the power delivery and detuning circuit is available in [43].

3.1 Introduction

The rapid growth of IoT devices has led to a corresponding growth in the adoption of near-field wireless charging for various applications [35, 38, 81, 86, 98, 109]. However, as the number of wireless power receivers grows, so will the number of chargers that might be counterfeit or not strictly standards-compliant [2, 3]. Given the critical nature of the tasks performed by IoT devices, protecting them from harsh transients imposed by counterfeit wireless chargers [61] is important. These transients could have potentially destructive impacts on both the receiver’s electronics and the battery being charged [49, 159]. This problem is made more challenging by the fact that the underdamped LC resonant tanks used by most resonant Wireless Power Transfer (WPT) systems [14, 94] tend to cause overvoltage or overcurrent conditions in response to the transients imposed by the charger.
A number of examples in literature [46, 52, 85, 135, 157] deal with authenticating a wired charger for the reasons of battery safety. Secure Hash Algorithm (SHA)-based cryptographic authentication protocols have been implemented commercially [1] for the same purpose. These solutions use a cryptographic element attached to the receiver that generates a challenge using a predetermined key. A genuine charger that has the appropriate key can then decrypt and respond to that challenge. The receiver is open-circuited until it receives the correct response, upon which it begins drawing energy from the charger. While a similar challenge-response protocol for charger authentication could be employed for incorporating secure charging into WPT, the projected scale of IoT wireless power receivers in the near future would make authentication based on a pre-shared secret (symmetric key), which is well suited for one charger-one receiver scenarios, unsustainable. Symmetric key authentication between the receiver and the charger requires that the receiver either be pre-programmed with the private keys of all possible chargers or be capable of exchanging a new key upon encountering a new charger. The former is clearly not scalable, while the latter requires all chargers and receivers share a master key that facilitates the key exchange over the same communication channel, thus introducing a new weak point in the system [20]. Both approaches require a secure memory on the receiver which cannot be read by an attacker, otherwise an attacker could extract these keys and impersonate a valid charger.

Instead, public key authentication uses two separate keys – a publicly known key used by the receiver for generating the challenge (public key) and its associated private key that is known only to the charger and is used for generating the response. The distribution of the charger public keys can be handled by issuing certificates signed by a trusted Certificate Authority (CA), in a way similar to the key-exchange handshake implemented in the TLS protocol. This avoids the need for implementing both secure key exchange and storage.

In a scenario where multiple receivers are coupled to the same charger, the power delivered to a receiver is a strong function of its proximity and orientation (which is related to the magnetic coupling coefficient) with respect to the charger coil [32, 33],
Figure 3-1: System architecture of a WPT system with multiple receivers connected to a single authenticated charger with more power going to the closer receiver. This physically imposed constraint might not necessarily reflect the actual energy requirements of the various receivers. This can be seen in the system-level diagram shown in Figure 3-1. In order to address these two disparate issues, two capabilities are required on the receiver side – the ability to (a) safely block power transmitted by a counterfeit wireless charger, and (b) be invisible to the charger so that more power can be delivered to a more distant receiver that might need it.

Section 3.2 describes a public key-based scheme for charger authentication for scalability and to avoid the need for secure key storage on the receiver. Measurement results for the complete system are presented in Section 3.3.

3.2 Authentication Block

The receiver authentication block (shown in Figure 3-2) decides whether to accept or block power from the charger based on the result of a challenge-response authentication protocol it executes with the charger. The authentication block consists of a baseband Pulse Width Modulation (PWM) modem for communication with the charger, an Serial Peripheral Interface (SPI) core for initial configuration and two-cryptographic accelerators: a Keccak based Sponge-Psuedorandom Number Genera-
A pseudo-random number generator (PRNG) similar to the one from Chapter 2 is used to generate random challenges and a National Institute of Standards and Technology (NIST) K-163 Elliptic Curve Scalar Multiplication (ECSM) that is used to verify the received responses using the charger's public key stored on the device. At start-up a fresh seed is used to configure the PRNG to ensure that no challenges are repeated.

The receiver communicates with the charger over the inductive link. Packets from the charger are modulated using On-Off Keying (OOK), while packets from the receiver use load-shift keying, both using the PWM baseband waveforms at 20 kbps. The use of PWM bits makes clock recovery unnecessary and simplifies demodulation at both ends.

### 3.2.1 Authentication Protocol

The challenge-response protocol employs a Diffie-Hellman based scheme, where the receiver generates a random scalar $c$ and sends the point $C = c \cdot G$ (where $G$ is the generator of the curve) as the challenge. The charger then uses its private key, $p$, to compute the response $R = p \cdot C$. The receiver then uses the charger's public key
\[ P = p \cdot G \] to check whether \( R = c \cdot P \). Thus the receiver needs to perform two scalar multiplications in order to authenticate the charger: first to generate the challenge and then to verify the response. Since scalar multiplications are a relatively costly operation, a dedicated ECSM unit is used to accelerate them.

### 3.2.2 ECSM Architecture Selection

From an implementation perspective, an ECSM designer must make many choices such as the choice of the curve, form of the co-ordinates, the field representation and the field operation implementation. Our ECSM implements scalar multiplication on the NIST K-163 curve [60] which is specified over the binary field \( \mathbb{F}_{2^{163}} \) and provides an 80-bit security level. Koblitz curves were chosen because they allow for efficient scalar multiplication by using the Frobenius endomorphism \( \tau \) as detailed in [142]. Scalar multiplication is commonly computed using a double and add ladder, but on Koblitz curves one can recode the scalar to its \( \tau \)-adic representation and replace the doubling by more efficient \( \tau(P) = \tau(x, y) = (x^2, y^2) \) operations. We use a normal representation for the binary field as this makes raising to a power of 2 i.e. \( x^{2^n} \) a left shift by \( n \)-bits, thus making the \( \tau \) operation essentially free. This is the main source of the efficiency of Koblitz curves, making it a very popular for low resource implementations [13,141]. Typically one needs to implement additional hardware for converting the scalar to the tauadic form. However, since we are only interested in ECSM for Diffie-Hellman operations we can simply interpret the CS-PRNG output as the recoded scalar and thus avoid the need for a separate converter.

The formulae for point addition on the curve are closely coupled with the chosen point representation. If we represent the field inversions and multiplication by \( I \) and \( M \) respectively, then the affine formulae require \( 2M \) and \( 1I \) while the projective Lopez-Dahab (LD) co-ordinates require \( 8M \) for mixed point addition [22]. For \( \mathbb{F}_{2^{163}} \) we can use Itoh-Tsuji (IT) inversion and perform an inversion using \( 9M \). Thus, it seems that the LD co-ordinates offer lower complexity when compared with the affine (\( 8M \) vs. \( 11M \)) at the cost of increased storage of 1 extra field element. Field multiplication in the normal basis is typically more complicated than the one in
the polynomial basis and from the point of compact implementation we use the bit-level Serial Input Parallel Output (SIPO) and Parallel Input Parallel Output (PIPO) multipliers from [26] and [127] respectively.

In order to better understand the impact of these design-choices, we implemented an affine SIPO ECSM and LD co-ordinate ECSMs using both the SIPO and SIPO multipliers. The various architectures are shown in detail in Figure 3-3. We implemented the affine SIPO architecture from [13] as a baseline in order to allow a fair comparison in our technology.

Figure 3-3: Comparison of K-163 ECSM Architectures

In all designs, the input point $P$ is stored in $x$ and $y$ and is never disturbed. The input scalar is stored in $p$ and the output point is retrieved from $x_1$ and $y_1$. We perform left-to-right double-and-add and store the intermediate points in $x_1$ and $y_1$. $z_1$ is used in LD co-ordinates for the third co-ordinate of the intermediate points. The field addition of any two registers is performed over two cycles: first write to $z$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Area (GE)</td>
<td>11340</td>
<td><strong>11100</strong></td>
<td>12350</td>
<td>11990</td>
</tr>
<tr>
<td>Latency (kcycles)</td>
<td>106.7</td>
<td><strong>98</strong></td>
<td>73.5</td>
<td>73.5</td>
</tr>
<tr>
<td>Gate Efficiency (ECSM/s/GE)</td>
<td>0.165</td>
<td><strong>0.184</strong></td>
<td>0.220</td>
<td>0.227</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>15.8</td>
<td><strong>12.7</strong></td>
<td>21.07</td>
<td>19.35</td>
</tr>
<tr>
<td>Energy (μJ/ECSM)</td>
<td>8.42</td>
<td><strong>6.22</strong></td>
<td>7.75</td>
<td>7.11</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of low-resource ECSM architectures

by XORing with 0 and then XOR z with the second argument. The inputs to the multiplier are always fixed to avoid any mux overhead. We first copy one argument to t and then we can multiply it with either $x_1$ or $z_1$ while accumulating the result in $z$ in a serial fashion.

An analysis of both the Affine and LD formulae shows that we can implement the necessary field operations using a single accumulator $z$. Our affine design strategically stores the intermediate results in $t$ itself in order to reduce the register count. IT inversion additionally requires the multiplication of $t$ by shifted versions of itself. This would require an additional register or a barrel shifter. We resolve this issue by using a SIPO multiplier. Hence we only need a 1-bit wide multiplexer that taps the appropriate bits from $t$ instead of a full barrel shifter. This also reduces the multiplexer complexity compared to the baseline design. When using LD co-ordinates we only need to perform a single final inversion of $z_1$. We can copy $z_1$ to $t$ and use $z_1$ for the shifted versions, thus avoiding an additional register for both the SIPO and SIPO multipliers.

### 3.2.3 ECSM Implementation and Performance

All the above architectures were synthesized in a 0.18 μm CMOS technology at a frequency of 200 MHz using regular $V_t$ standard cells with a nominal supply voltage of 1.8 V. Table 3.1 compares these synthesized designs. Note that the baseline architecture was re-implemented in our process technology to offer a fair comparison.

From our results we observe that the LD formulae (using $8M$) when compared
with the Affine (11M) formulae result in better latency and gate-efficiency for the LD implementations. However, surprisingly this does not translate to better energy-efficiency due to the larger mux overhead and higher load seen by z. The SIPO multiplier is larger than the SIPO multiplier and hence the LD SIPO implementation is worse on all metrics when compared with the LD SIPO implementation. For the affine implementations we observe that although the SIPO multiplier is larger the reduction in the register count and mux complexity results in the overall smallest implementation with roughly 27% lower energy when compared with the baseline implementation. Based on this analysis the Affine SIPO architecture was chosen for our implementation.

Since the overall authentication protocol requires just two ECSMs at the verifier we can use aggressive voltage scaling to improve the energy efficiency while maintaining real-time operation. Figure 3-4 reports the performance of the ECSM as function of the supply voltage. The use of only registers for the entire state (avoiding the use of SRAM) allows us to scale the voltage down to 0.475 V. We observe that at the minimal energy point (0.475 V), the ECSM requires 0.77 µJ per ECSM. A comparison of the implemented ECSM vs. recently reported works is presented in Table 3.2.
Table 3.2: Comparison of recent low-resource ECSM implementations

<table>
<thead>
<tr>
<th></th>
<th>This work</th>
<th>Roy et al. [141]&lt;sup&gt;a&lt;/sup&gt;</th>
<th>Azaderakhsh et al. [13]&lt;sup&gt;a&lt;/sup&gt;</th>
<th>Lee et al. [96]&lt;sup&gt;a&lt;/sup&gt;</th>
<th>Pessl et al. [12]&lt;sup&gt;b&lt;/sup&gt;</th>
<th>Wenger et al. [155]&lt;sup&gt;a&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>0.18 µm</td>
<td>0.13 µm</td>
<td>65 nm</td>
<td>90 nm</td>
<td>0.13 µm</td>
<td>0.13 µm</td>
</tr>
<tr>
<td>Curve type</td>
<td>Koblitz</td>
<td>Koblitz</td>
<td>Koblitz</td>
<td>Binary</td>
<td>Prime</td>
<td>Binary</td>
</tr>
<tr>
<td>Curve size</td>
<td>163</td>
<td>283</td>
<td>163</td>
<td>160</td>
<td>160</td>
<td>163</td>
</tr>
<tr>
<td>Latency (k-cycles)</td>
<td>98</td>
<td>1566</td>
<td>106.7</td>
<td>62.5</td>
<td>139.9</td>
<td>341.8</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
<td>1</td>
<td>16</td>
<td>13.5</td>
<td>204</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Area (GE)</td>
<td>11471</td>
<td>10204&lt;sup&gt;b&lt;/sup&gt;</td>
<td>11571</td>
<td>98000</td>
<td>12448</td>
<td>11778</td>
</tr>
<tr>
<td>Power (µW)</td>
<td>7.93</td>
<td>97.70</td>
<td>77.2</td>
<td>-</td>
<td>42.42</td>
<td>63.3</td>
</tr>
<tr>
<td>Energy (µJ/ECSM)</td>
<td>0.77</td>
<td>9.56</td>
<td>0.61</td>
<td>9.3</td>
<td>-</td>
<td>21.6</td>
</tr>
</tbody>
</table>

<sup>a</sup> Only post-synthesis results available  
<sup>b</sup> Including RAM for storing point coordinates  
<sup>c</sup> ECSM core including 823-GE hash engine, exact ECSM energy unavailable

3.3 Measurement Results

3.3.1 Test Setup

The receiver was fabricated in a 0.18 µm CMOS process with 5 V transistors and occupies 2.8 mm² active area. The addition of the auxiliary rectifier and its associated control circuits increases the active area of the IC by approximately 35%. The charger was implemented using an H-bridge push-pull Power Amplifier (PA) made up of off-the-shelf components integrated on a PCB and driven at $f_{op}$ (6.78 MHz). The charger coil is an 8-turn, 3-in.×2-in. rectangular spiral made of PCB traces with 4.3 µH measured inductance. The PA drives a series LC tank comprising the transmit coil and its series capacitor tuned at $f_{op}$. The digital baseband functionality on the charger that generates responses to the challenges sent by the receivers is implemented on an FPGA.

The peak dc output power in the tuned mode is 520 mW and the peak end-to-end efficiency ($P_{out,dc}/P_{in,dc}$) is 74%. The control circuits on the receiver consume 32 mW of power. This is with the charger operating from a 4-V supply and the receiver delivering power to a 4-V dc voltage source (both consistent with cell voltages of a standard Li-ion battery). The charger and receiver are aligned center-to-center with a vertical distance of 0.5 in. between them. A die micrograph and performance
Figure 3-5: Die micrograph and performance summary of implemented receiver

summary of the implemented receiver is shown in Figure 3-5. The measurement setup for testing both charger authentication and co-operative power balancing is shown in Figure 3-6. For the power balancing measurement setup, the two receivers are held at different distances from the same charger by using nylon board spacers of different lengths.

3.3.2 Detuning Mechanism

We now briefly describe the detuning mechanism that enables rejection of power. A more complete description can be found in the Nachiket Desai’s thesis [44]. At a high-level the amount of power-rejected by the receiver can be controlled employing duty-cycle control of a small auxiliary rectifier. The duty-cycle control modulates the input resistance of the rectifier. This in turn controls the current in the auxiliary coil and causes it to detune.
Figure 3-6: Measurement setup for charger authentication and power balancing

3.3.3 Charger Authentication

The authentication flow and the corresponding measured waveforms are shown in Figure 3-7. The receiver starts up in the fully detuned state with the auxiliary rectifier enabled and waits for a charger to come in range. When the envelope detector detects a charger, the authentication block on the receiver generates a challenge. Upon receiving the challenge, the charger uses its private key to generate a response which is then checked by the receiver. Upon successful authentication, the receiver disables the auxiliary rectifier to commence resonant power transfer. This causes $P_{out,dc}$ to rise to 128 mW from the 8 mW detuned value for the case shown in Figure 3-7.

3.3.4 Dynamic Power Balancing

Figure 3-8 shows the system performance when two receivers are coupled to the same charging coil with different coupling coefficients. If the receivers individually try to maximize their output powers, the power delivered is skewed heavily in favor of
Figure 3-7: Authentication flowchart and transmitter and receiver traces during authentication.
Co-operative Detuning to Balance Output Power

Figure 3-8: Co-operative power sharing by receivers at different distances from the charger ($d_{z1}$ and $d_{z2}$ are the distances to the closer and farther receivers respectively)

the nearby receiver. Detuning the nearby receiver can reduce or even reverse the physically imposed asymmetry in the delivered power, and distribute power based on the relative needs of both receivers. Since the charging coil current is limited by the nearby receiver, detuning it allows the charger output power to rise. This induces a larger EMF on the farther receiver and allows more power to be delivered to it. Figure 3-8 shows power numbers for two distance configurations when the receivers don't co-operate and when they co-operate with the goal of equalizing their individual dc output powers for two separate levels of coupling asymmetry.

Figure 3-9 shows that maximally detuning the closer receiver allows the asymmetry in power delivery to be reversed even when the two receivers are at distances in a 4:1 ratio away from the charger. In both cases, the total efficiency ($\eta_{total}$) is highest when the receivers do not co-operate because most of the power goes to the closer receiver at higher efficiency. By detuning the closer receiver, a more significant fraction of the power can be delivered to the farther receiver at lower $\eta_{total}$. Thus, detuning allows for a trade-off between $\eta_{total}$ and balanced power delivery.

From Figure 3-9, it can also be seen that partially detuning the closer receiver using
d_1 = 0.25 in.  
d_2 = 1 in.

1350x

\[ d_{z1} = 0.5 \text{ in.}  \]
\[ d_{z2} = 0.75 \text{ in.}  \]

3x

1290x

Figure 3-9: Range of power ratios obtainable by co-operative action by the receivers.

Duty cycle control allows the system to achieve intermediate power distribution ratios. This can offer an overall efficiency benefit when compared against time multiplexing the system between the closer receiver being maximally detuned and fully tuned. For example, in the 4:1 asymmetric case shown in Figure 3-9, the overall efficiency in achieving a 50-50 energy split between the two receivers is about 42% with partial detuning of the closer receiver. If instead, time multiplexing is chosen to achieve the same energy split, the system must spend 82% of its time with the closer receiver maximally detuned (a state with lower overall efficiency). This leads to a time-averaged overall efficiency of 37%.

### 3.4 Conclusion

IoT devices with wireless charging capability need to be protected from damaging transients imposed by counterfeit chargers to ensure safe operation. Moreover, with multiple receivers coupled to the same charging coil with different coupling coefficients, the amount of received power is heavily dictated by the physical constraints instead of the actual power requirements of the receivers. This chapter presented a wireless power receiver that addressed both these issues by controlling the amount by which the receiver is detuned from the frequency at which the charger transmits...
power. The charger is authenticated using a public key-based authentication scheme that is scalable and avoids the need for the receiver to securely store a private key. We demonstrated a 10k NAND gate ECC accelerator that requires just 0.77μJ per ECSM. The receiver detuning technique achieves up to 16× power blocking and is capable of completely inverting the power distribution profile in a one charger-two receiver scenario with 4:1 distance asymmetry between the receivers.
Chapter 4

Practical Leakage Resilient Identification Schemes

4.1 Introduction

With the proliferation of a large number of IoT devices, their security has become an important issue. In many applications, such as RFID tagging in the supply-chain and access control smart-cards, authentication is a key cryptographic primitive that is required to be implemented. Often the complexity of key-distribution in embedded environments necessitates a public-key solution [20]. In the presence of severe resource constraints ECC based ID schemes [138] are a very attractive option due to their small key-sizes which lead to low-area implementation [43, 80].

On the flip-side, this very embedded nature allows attackers physical access and often makes them susceptible to implementation attacks such as side-channel attacks [64, 102]. In this context, the security of ECC implementations against side-channel attacks gains added significance. While various implementation level countermeasures such the use of complete formulae [31], constant time ladders [111], scalar splitting [39] and scalar randomization [40] have been proposed to counter side-channel attacks, an ever sophisticated arsenal of attacks have also been developed against these countermeasures. For example complete formulae and ladders are extremely effective against SPA but susceptible to DPA. Scalar splitting and random-
ization are effective against DPA but susceptible to template attacks [34,104]. Even in the absence of template attacks DPA countermeasures need to be analyzed with great care. Recent attacks [150] demonstrate that even a few bits of bias in the scalar randomization (as low as a single bit [12]) can be exploited by an attacker if multiple power traces with the same scalar can be recorded.

As such there is a clear need to marry implementation level countermeasures with inherently leakage-resilient cryptographic algorithms [51,106]. Instead of viewing cryptographic algorithms as simple black-boxes, leakage-resilient cryptography accepts that certain traditionally secret parts of the algorithm will be available to the attacker and aims to ensure the security of these algorithms in the presence of such leakage. A common design pattern in many leakage-resilient cryptographic algorithms is the notion that any secret-key material used in the algorithms is frequently refreshed in order to avoid leaking the entire key. Although this heuristic was first proposed informally [91], since then many provably leakage-resilient primitives including Pseudorandom Generators (PRGs) [51] and Pseudorandom functions (PRFs) [119] have been developed. In fact even leakage-resilient public-key primitives are possible [30,97]. However very few [62] concrete implementations of these primitives are available to benchmark their suitability to embedded environments.

This section aims to fill this very gap in the context of leakage resilient ID schemes. More concretely we raise the following question: “How can one implement an efficient leakage-resilient ID scheme that allows a resource constrained prover to identify itself to a remote verifier without the requiring the verifier to maintain any synchronized state?”

In order to further motivate the need for a new leakage resilient ID scheme let us first recap our two previous solutions. Although, the protocol from section 2.1 is leakage resilient, it is inherently a stateful symmetric scheme that relies on synchronized shared secret keys [86]. This poses three major-drawbacks:

1. Since the verifier contains secret keys of all potential provers they are lucrative targets for attack.
2. In a multi-verifier scenario, we need to a mechanism to securely synchronize the verifier across all the verifiers.

3. All verifiers can potentially spoof the prover's identity to other verifiers.

ID schemes that rely on the public-key cryptography, such as one from section 3.2, do not suffer from any of these drawbacks but require extensive implementation level counter-measures [31,39,40,111] to protect against side-channel attacks.

Thus the goal of this work is to marry the best features of both these schemes and develop a leakage-resilient ID scheme where the entire verifier state is public and static. At a high-level our construction can be visualized as a challenge-response scheme, where the prover uses their secret key to generate responses to the verifier's challenges. The verifier uses the prover's public key to authenticate the prover's response. In order to gain leakage resilience we allow the prover to update their secret as frequently as they desire subject to the constraint that the original public key can still validate responses generated by these fresh secret keys. As a result the verifiers' databases would simply contain the static public keys all the provers and would neither need to be secure nor synchronized across verifiers, while allowing the prover to maintain leakage resilience.

4.1.1 Threat Model

The threat model that we consider in this work is almost exactly the same as that from Chapter 2. In particular, explicitly consider:

- Passive side-channel attacks: The attacker may passively monitor the power and/or the EM emission from the prover in preparation for a DPA or DEMA respectively.

- Protocol attacks: The protocol must necessarily be secure in the absence of leakage. Thus the attacker must not be able to impersonate a valid prover even if he can capture, corrupt or replay arbitrary messages between the prover and verifier.
4.2 Preliminaries

We use the following notation in what follows. We represent vectors using lowercase boldface (i.e. \(x \in \mathbb{Z}_q^n\)) and matrices using uppercase boldface (i.e. \(X \in \mathbb{Z}_q^{n \times m}\)). Similarly we use the shorthand notation \(g^x\) and \(g^X\) to represent vector and matrices in the exponent, i.e. \((g^x)_{i,j} = g^{x_{i,j}}\). Given a domain \(D\) and a distribution \(\mathcal{D}\) we write \(x \sim \mathcal{D}\) to denote sampling \(x\) over \(X\) according to the distribution \(\mathcal{D}\). We use the shorthand notation \(x \sim X\) to denote sampling over \(X\) according to the uniform distribution. Finally we indicate two samples \(X \sim X\) and \(Y \sim Y\) are statistically-indistinguishable using the notation \(X \approx_s Y\).

4.2.1 Leakage Assumptions

Our proof of leakage resilience follows from a line of results [5, 30, 47, 97] on leakage resilience that show that vectors from the column span of a random matrix \(A\) are statistically indistinguishable from uniformly random vectors. The exact formulation we invoke comes from [5].

**Lemma 4.2.1.** *(Affine Subspace Hiding)* Given \(n \geq d \geq u > 0\), let \(A \sim \mathcal{G}_q^{n \times d}\), \(V \sim \mathcal{G}_q^{d \times u}\) and \(B \sim \mathcal{G}_q^{n \times u}\) and \(\text{Leak}(\cdot)\) be a leakage function whose output is bounded by \(L\)-bits. Then we have,

\[
(\text{Leak}(AV + B), A) \approx_s (\text{Leak}(U), A)
\]

subject to \((d - u) \log(q) - L = \omega(\log(\lambda)), n = \text{poly}(\lambda)\) and \(q = \lambda^{\omega(1)}\).

4.2.2 Bilinear Groups and Computational Assumptions

In what follows we work with affine groups \(\mathcal{G}\) and use multiplicative notation to represent these groups. Let \(\mathcal{G}_1, \mathcal{G}_2\) and \(\mathcal{G}_T\) be three affine groups of prime-order \(q\) with the corresponding generators \(g_1, g_2\) and \(g_T\). A bilinear map is a function \(e : \mathcal{G}_1 \times \mathcal{G}_2 \rightarrow \mathcal{G}_T\) s.t. \(\forall x_1 \in \mathcal{G}_1; x_2 \in \mathcal{G}_2; a, b \in \mathbb{Z}_q; e(x_1^a, x_2^b) = e(x_1, x_2)^{ab}\) (bilinearity)
and \( c(g_1, g_2) \neq 1 \) (non-degeneracy). A non-degenerate bilinear map is often referred to as a pairing.

**Decisional Diffie-Hellman (DDH) assumption**

The DDH assumption states that given a group \( G \) with generator \( g \) and \( a, b, c \leftarrow \mathbb{Z}_q \) the two tuples \((g, g^a, g^b, g^{ab})\) and \((g, g^a, g^b, g^c)\) are computationally indistinguishable.

**Matrix Linear assumption**

The matrix linear assumption [115] states that given a group \( G \) with generator \( g \) and matrices \( X_1, X_2 \leftarrow \mathbb{Z}_q^{m \times l} \) with rank \( r_1 \) and \( r_2 \), s.t. \( m, l \geq 3, r_2 \geq r_1 \geq 2 \) the two matrices \( g^{X_1} \) and \( g^{X_2} \) are computationally indistinguishable. Note that the matrix linear assumption is weaker than the DDH assumption, i.e. the hardness of DDH implies the hardness of the matrix linear assumption.

**Symmetric External Diffie-Hellman (SXDH) assumption**

Given a tuple of groups \((G_1, G_2, G_T)\) that admit a paring, the SXDH assumption states that the DDH assumption holds in both \( G_1 \) and \( G_2 \).

**4.2.3 Barreto-Naehrig curves**

BN curves [19] are a widely adopted candidate construction for bilinear groups that where SXDH is believed to be hard. BN curves are prime-order curves defined over the field \( \mathbb{F}_p \), satisfying the equation \( E : y^2 = x^3 + b \). The prime \( p \) and the curve-order \( q \) are parametrized by \( u \in \mathbb{Z} \) by the following formulae: \( p = 36u^4 + 36u^3 + 24u^2 + 6u + 1 \) and \( q = 36u^4 + 36u^3 + 18u^2 + 6u + 1 \). We implement the BN254 curve specified by \( u = -(2^{62} + 2^{55} + 1) \) and \( b = 2 \) which offers 100bit security in the light of recent advances in the number field sieve algorithm [18]. The three groups \( G_1, G_2, G_T \) are given by \( E(\mathbb{F}_p)[q] \), \( E'(\mathbb{F}_p)[q] \) (where \( E' \) is a sextic-twist of \( E \)) and the sub-group \( \mu_q \subset \mathbb{F}_{p^{12}} \) of the \( q \)-th roots of unity in \( \mathbb{F}_{p^{12}} \). The pairing function we compute is the optimal-ate pairing as specified in [114]
4.2.4 Identification Scheme

An ID scheme is an interactive protocol, that allows a prover $P$ to convince a verifier $V$ that it knows a witness $sk$, corresponding to a public instance $pk$. In particular we focus on the three-round protocols ($\Sigma$-protocols).

**Definition 4.2.1.** An ID scheme is a three-round interactive protocol between $P$ and $V$ which consists of the following five probabilistic polynomial time (PPT) algorithms ($Gen$, $Commit$, $Chal$, $Resp$ and $Verify$) defined as follows:

- $(sk, pk) \leftarrow Gen(1^\lambda)$ is a PPT algorithm that takes as input the security parameter and generates the witness and a public instance.

- $u \leftarrow Commit(pk)$ is a PPT algorithm that generates the prover's first message $u$ based on the public instance.

- $c \leftarrow Chal(pk)$ is a PPT algorithm that the verifier uses to generate a challenge for the prover based on the public instance.

- $r \leftarrow Resp(c, sk, pk)$ is a PPT algorithm that the prover uses to respond to the verifier's challenge, the public instance and it's witness.

- $0, 1 \leftarrow Verify(u, c, r, pk)$ is a deterministic polynomial time algorithm that the verifier uses to decide whether to accept the prover's response ('1' indicates accept and '0' indicates reject).

Additionally we require the following three properties of our ID scheme: correctness, soundness and (honest-verifier) zero-knowledge given as follows:

**Definition 4.2.2.** (Correctness) $\forall (sk, pk) \leftarrow Gen(1^\lambda)$,

$$Pr[Verify(u, c, r, pk) = 0] \leq negl(\lambda)$$

where the probability is taken over the random coins of all the algorithms.
Definition 4.2.3. (Soundness) There exists a PPT algorithm EXT s.t. for all pairs of transcripts $(u, c_1, r_1)$ and $(u, c_2, r_2)$ that are accepted by Verify,

$$Pr[\text{EXT}(u, c_1, r_1, c_2, r_2, pk) = sk] \geq \text{nonnegl}(\lambda)$$

where the probability is taken over the random coins of EXT.

Definition 4.2.4. (Honest Verifier Zero Knowledge) There exists a PPT algorithm SIM(pk) that generates transcripts $(u, c, r)$ that are distributed identical to the transcripts generated interactively.

4.2.5 Schnorr ID

We briefly recall the classic Schnorr (honest-verifier) ID scheme [138].

Definition 4.2.5. (Schnorr ID) Given a group $\mathbb{G}$ where discrete-log is hard, the Schnorr ID scheme is described by the following tuple. Let $q = |\mathbb{G}|

- Gen(1^\lambda) : s \leftarrow Z_q^*, \text{Output } (s, g^s)
- Commit(g, \mathbb{G}) : u \leftarrow Z_q^*, \text{Output } g^u
- Chal(g, \mathbb{G}) : c \leftarrow Z_q^*, \text{Output } c
- Resp(c, s, u, g, \mathbb{G}) : \text{Output } r \leftarrow u + sc.
- Verify(g^u, c, r, g^s) : \text{Accept iff., } r = g^u(g^s)^c

Theorem 4.2.2. Schnorr ID is correct, sound and honest-verifier zero knowledge.

4.3 Protocol

While the classic Schnorr ID scheme is correct, sound and honest-verifier zero-knowledge, it is not secure against continual leakage even under the only-computation leaks assumption. This result holds true even with a strong upper bound on the rate of leakage (say at most a single bit of leakage). One way to see this is to use a leakage
function that leaks a different bit of the secret in every iteration of the protocol. One approach to solve this issue is to re-randomize the scalar in a way that does not require a corresponding change in the public key. A second constraint however is that the re-randomization itself is secure against continual leakage. We now propose a candidate construction with these properties.

### 4.3.1 Continual Leakage Resilient ID

We start our construction by considering a vector version of the Schnorr scheme over the bilinear groups $G_1$, $G_2$ and $G_T$. We promote the scalars in the original scheme to elements of $G_1$ and the verification check to the corresponding pairing check. Using this vector version of the scheme allows us to pick multiple secret keys that correspond to the same public key. The key update mechanism then switches between these various valid secret keys. The correctness, soundness and honest-verifier zero-knowledge properties of this scheme follow from the correspondingly modified versions of the proofs for the Schnorr scheme. We finally use a proof strategy similar [30] to show that our key-updates are indeed leakage resilient.

**Definition 4.3.1.** (Continual Leakage Resilient ID) Given groups $G_1$, $G_2$ and $G_T$ of prime-order $q$, which admit a non-degenerate bilinear pairing $e$ and where the SXDH problem is hard, the continual leakage resilient ID scheme is described by the following set of PPT algorithms.

- **Gen:** $x, y, a \leftarrow \mathbb{Z}_q^n, s.t. \langle a \cdot x \rangle = 0$ and $\langle a \cdot y \rangle = b$, output $sk = (g_1^x, g_1^y)$ and $pk = (g_2^a, g_1^b)$

- **Commit:** $u \leftarrow \mathbb{Z}_q^n$, output $g_t^s$ s.t.

$$g_t^s = \prod_{i=1}^{n} e(g_1^{u_i}, pk_{0,i})$$

- **Chal:** $c \leftarrow \mathbb{Z}_q$, output $c^{-1}$

- **Resp:** Output $g_1^r = (g_1^u)^c \cdot sk_1$. 

68
- **Verify**: Accept iff.,

\[ \prod_{i=1}^{n} e(g_1^{e_{i}c^{-1}}, pk_{0,i}) = g_i^{e} \cdot e(pk_i^{-1}, g_2) \]

- **Update**: \( k_0, k_1 \leftarrow \mathbb{Z}_q \). Output \( sk = (sk_0^{k_0}, sk_1 \cdot sk_0^{k_1}) \)

The main idea behind the key-update step is to maintain the following invariant if \( sk' = (g_1^{x'}, g_1^{y'}) \) then \( (a \cdot x') = 0 \) and \( (b \cdot y') = b \). The key-update step ensures this fact by only scaling the first component of the secret-key and then adding it to the second component on each update step.

**Lemma 4.3.1.** The scheme 4.3.1 is correct.

The correctness of the scheme follows from a simple substitution of variables in the verification equation and using the two secret-key invariants.

**Lemma 4.3.2.** The scheme 4.3.1 is sound.

To prove the soundness of the scheme we need to construct an efficient extractor given the two transcripts \( T_a = (g_1^s, c_a, (g_1^n)^{c_a} \cdot sk_1) \) and \( T_b = (g_1^s, c_b, (g_1^n)^{c_b} \cdot sk_1) \). This gives us a direct extractor for \( sk_1 = (\frac{T_a[2]^{c_a-1}}{T_b[2]^{c_b-1}})^{-1} \). Repeating this for two instances of \( sk_1 \) allows us to also extract \( sk_0 \).

**Lemma 4.3.3.** The scheme 4.3.1 is honest-verifier zero-knowledge.

The honest verifier zero-knowledge property follows from the analogous proof for Schnorr ID. Given a random response \( g_1^s \) and challenge \( c \) we can simulate the commitment message as follows: \( g_1^s = e(pk_1^c, g_2) \cdot \prod_{i=1}^{n} e((g_1^{u_i})^{c_{i-1}}, pk_{0,i}) \).

This bring us to the final property we desire: leakage resilience. Following the analysis from [30] we see that our scheme as described can tolerate a leakage of \( 1 - \frac{2}{n} \) bits between key-updates and a modest \( \log(\lambda)/(n \log(p)) \) bits of leakage during the key-update. We simplify our presentation by ignoring any leakage during the initial key generation process. We note that this is a very practical assumption for RFID
tags, since it is unlikely for the adversary to be present when these tags are initially provisioned.

The proof of this property follows the three step procedure: first we show that the scheme tolerates leakage from a single run of the protocol, next we show that the scheme tolerates continual leakage in the absence of leakage during the key-update phase and finally we show that the scheme is continual leakage resilient in the presence of leakage during key update.

At a high-level our goal is to show that under the affine subspace hiding lemma, there is exists an extractor for an adversary $A$ that can generate responses to arbitrary challenges in the presence of leakage. This follows because, given the public key one can generate transcripts from the correct distribution as seen in the honest-verifier zero-knowledge property. Additionally one now also needs to simulate the leakage queries corresponding these transcripts. We can simply provide leakage on a random vector $g^{r \cdot z + y}$ where $\langle a \cdot z \rangle = 0$ and $\langle x \cdot z \rangle = 0$ instead of the secret-key $sk_1$. No adversary $A$ can distinguish this random leakage from leakage from the correct key under the affine subspace hiding lemma. Thus if the adversary can successfully generate a valid response we can run our original extractor from the proof of soundness. The proof of the last two steps is exactly the same as in [30].

4.3.2 Security Guarantees, Computation Requirements and Practical Issues

Next we consider a few practical issues related to the performance and side-channel leakage. The computational and communication requirements for implementing the scheme in terms of group operations and base-field elements respectively are presented in Table 4.1. Our description of the protocol preferentially pushes all the exponentiations to the base group $G_1$ to minimize the computational overhead. We note that typical use cases for scheme will involve a resource constrained prover who needs to the implement the Commit, Resp and Update operations. We note that since the commit message requires a product of $n$-pairings, we can optimize this computation
Table 4.1: Operation Counts and Communication Cost

<table>
<thead>
<tr>
<th>Operation</th>
<th>$G_1$ Exp.</th>
<th>Pairing</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>$n$</td>
<td>$n$</td>
<td>12</td>
</tr>
<tr>
<td>Chal</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Resp</td>
<td>$n$</td>
<td>0</td>
<td>$n$</td>
</tr>
<tr>
<td>Verify</td>
<td>$n+1$</td>
<td>$n+1$</td>
<td>0</td>
</tr>
<tr>
<td>Update</td>
<td>$2n$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

by splitting the pairing computation into its constituent operations of Miller loop evaluation and the final exponentiation. We can use the commutative and associative properties in $F_{p^{12}}$ to replace the $n$-final exponentiations with a single exponentiation. As shown in section 5.9 the Miller loop and the final exponentiation take roughly the same amount of time and thus we can reduce the overall cost of the Commit operation to roughly $(n+1)/2$ Pairings.

Next we summarize the security implications of the above proof. Setting $n = 3$, the security proof guarantees even if $\frac{1}{3}$rd of the key is leaked during every iteration, the protocol remains secure. This seems to be a achievable goal since most practical DPA attacks require much higher than three traces for successful key-recovery. This allows a reasonable compromise between the implementation complexity and offered security. However an important point to note is that during the key-update itself only a logarithmic amount (few bits) of leakage can be tolerated. Thus we have two concrete recommendations for instantiating this protocol. If the user can guarantee the key-update happens only in a secure environment then the hardware needs to satisfy very mild assumptions and no DPA countermeasures are necessary. On the other hand if key-update must happen in adversarial environments then hardware DPA countermeasures are recommended. The implemented countermeasures must then be validated to ensure $\leq 1$ bits of leakage per trace so as to meet the leakage bound over the 6 ECSM required for the key-update.

Finally we note that in order to achieve the requisite leakage bound an implicit condition is that the implementation must be protected using SPA countermeasures. Failing this the entire key could be leaked in a single run of the protocol. This would
violate the assumptions of the proof and lead to prevent a complete-break of the system. A simple approach to achieve this goal is to delegate group operations to a constant time hardware implementation.

4.4 Hardware Accelerated Implementation

In this section, we describe the hardware-accelerated implementation of our proposed leakage-resilient ID protocol. It is important to note that instead of designing custom pairing accelerators, we have re-used a hardware platform [17], originally designed to accelerate the TLS protocol, to demonstrate the practicality of our proposed scheme. This chip was developed in collaboration with Utsav Banerjee and Computational Structures Group at MIT. We provide an overview of this chip, present the details of its cryptographic hardware accelerators and provide detailed benchmarks of pairing computations.

4.4.1 Chip Overview

Fig. 4-1 provides a high-level block diagram of the system, which consists of a Cryptographic Engine (CE) and a 3-stage RISCV processor [154] supporting the RV32I instruction set. The RISCV processor is based on the open-source Bluespec RISCV cores developed by the Computational Structures Group at MIT at [107]. The cryptographic engine, comprised of a protocol controller, a dedicated 2 KB RAM, and AES-128 (in ECB and GCM modes), SHA-256 (in message digest, HMAC and HMAC-DRBG modes) and prime field ECC primitives, accelerates the DTLS protocol. The RISCV processor has a 16 KB instruction cache and a 64 KB data memory. An SD card is used as the backing storage for larger programs. The cryptographic accelerators can be configured and accessed through a memory-mapped interface, using simple load and store instructions in RISCV software. Sleep mode is implemented on the RISCV, to save power, by gating its clock when cryptographic tasks are delegated to the CE. The CE uses a dedicated hardware interrupt to wake the processor on completion of these tasks. The CE is clocked by a software-controlled divider to
decouple the processor operating frequency from the long critical paths in the ECC accelerator.

![Diagram of RISCV processor](image)

**Figure 4-1: Overview of the RISCV processor**

The test chip, shown in Fig. 4-2, was fabricated in a 65 nm LP CMOS process, and supports voltage scaling from 1.2 V down to 0.8 V. At 0.8 V, the RISCV processor and the cryptographic engine can operate at maximum frequencies of 20 MHz and 16 MHz respectively. The RISCV processor occupies 34 kGE, and achieves 0.96 DMIPS/MHz, consuming 40.36 $\mu$W/MHz, when running the Dhrystone v2.1 benchmark. This is comparable to the Dhrystone performance of an ARM Cortex-M0+ processor. The cryptographic engine occupies 149 kGE, and uses 6.75KB of SRAM. Performance of the CE accelerators will be discussed in detail.
4.4.2 Energy-Efficient Reconfigurable ECC

The ECC accelerator described in this section was contributed by Utsav Banerjee and we refer the reader to his thesis [16] for a detailed description. We briefly summarize some salient features which are relevant to this application. Our ECC accelerator has been designed to support all short Weierstrass curves (of the form $y^2 = x^3 + ax + b$) over prime fields up to 256 bits. As shown in Fig. 4-3, the prime $p$, its bit-length ($\leq 256$), and the curve parameters $a$ and $b$ can be configured to allow this flexibility. For the BN254 curve, we have $a = 0$ and $b = 2$, and $p$ is a 254-bit prime. Two key components of our ECC hardware are the modular multiplier and the modular inverter, which are used to accelerate field arithmetic. They can also be accessed standalone, through software, to perform modular operations.

In order to support arbitrary primes, we have implemented multiplication with interleaved modular reduction [73], which is essentially a loop of "double-add-reduce". Three adders are used for this computation - one for addition and two for reduction. The interleaved reduction involves conditional subtractions, which are all performed...
in the same cycle, using adders and multiplexers. This ensures that our modular multiplication hardware runs in constant time, and there is no potential timing side-channel. While most conventional ECC designs choose 16-bit or 32-bit data-paths for modular arithmetic, we use 256-bit wide adders in our design. This reduces control circuitry and muxing logic, thus improving overall performance and energy-efficiency. The same hardware is re-used for modular addition.

Resource-constrained ECC implementations typically use projective coordinates to avoid modular inversion in the ECSM inner loop, at the cost of extra multiplications and a final expensive Fermat inversion. However, we have implemented a dedicated modular inverter based on the extended Euclidean algorithm. Coupled with the use of affine coordinates, this results in a 48% reduction in ECSM energy by trading off the extra multiplications for cheaper Euclidean inversions.

Figure 4-3: Block diagram of hardware implementation of prime field modular arithmetic and elliptic curve cryptography.

In order to prevent simple power analysis side-channel attacks on the hardware-
Table 4.2: \( \mathbb{F}_p \) operation counts in pairing computation

<table>
<thead>
<tr>
<th></th>
<th>Miller Loop</th>
<th>Final Exp</th>
<th>Pairing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbb{F}_p ) Add / Sub</td>
<td>21283</td>
<td>17659</td>
<td>38942</td>
</tr>
<tr>
<td>( \mathbb{F}_p ) Mul</td>
<td>9484</td>
<td>8481</td>
<td>17965</td>
</tr>
<tr>
<td>( \mathbb{F}_p ) Inv</td>
<td>70</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>Total Cycles</td>
<td>( \approx 9.9M )</td>
<td>( \approx 8.7M )</td>
<td>( \approx 18.6M )</td>
</tr>
</tbody>
</table>

accelerated ECSM, a zero-less signed digit representation [79] of the scalar \( k \) is used. Each hardware-accelerated SPA-secure ECSM operation on the BN254 curve takes \( \approx 495k \) cycles, which translates to 31 ms at a clock frequency of 16 MHz. The corresponding energy consumption is 16.72 \( \mu \)J at a supply voltage of 0.8 V.

4.4.3 Pairing Implementation

Our pairing library implements arithmetic over the fields \( \mathbb{F}_p, \mathbb{F}_{p^2}, \mathbb{F}_{p^6} \) and \( \mathbb{F}_{p^{12}} \), and over the elliptic curve groups \( E(\mathbb{F}_p) \) and \( E(\mathbb{F}_{p^2}) \). All field operations in \( \mathbb{F}_p \) and scalar multiplication in \( E(\mathbb{F}_p) \) are accelerated in hardware. They are used by the other field/group operations as underlying primitives, with the control implemented as RISCV software.

Table 4.2 enumerates the number of individual \( \mathbb{F}_p \) operations required in the Miller loop, the final exponentiation and the pairing computation as a whole. The total number of cycles required for performing each computation, on our hardware platform, is also presented. We observe that cycle counts for the Miller loop and the final exponentiation are approximately equal, thus motivating some of the protocol optimizations described in section 4.3. When compiled using the "-Os" flag in RISCV GCC, our pairing library has code size \( \approx 14.3 \) KB. Therefore, it can easily fit in the 16 KB instruction cache of the RISCV processor, and does not incur any cache miss penalties except for the first time the program gets loaded. Each pairing computation takes \( \approx 18.6M \) cycles, resulting in a run-time of 1.17 s at 16 MHz, with energy consumption 678 \( \mu \)J.
Table 4.3: $\mathbb{F}_p$ operation counts in elliptic curve point addition in affine coordinates

<table>
<thead>
<tr>
<th></th>
<th>Addition</th>
<th>Doubling</th>
<th>Complete Addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_p$ Add / Sub</td>
<td>8</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>$\mathbb{F}_p$ Mul</td>
<td>3</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>$\mathbb{F}_p$ Inv</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total Cycles</td>
<td>$\approx 4.4k$</td>
<td>$\approx 5.0k$</td>
<td>$\approx 9.3k$</td>
</tr>
</tbody>
</table>

4.4.4 Elliptic Curve Point Addition

Traditional elliptic curve point addition on short Weierstrass curves involves the "chord-tangent" formulae, with the "tangent" doubling formula being used when the two points are equal, and the "chord" addition formula being used otherwise. As a result point doubling (DBL) and point addition (ADD) computations have different cycle counts, thus making the traditional $E(\mathbb{F}_p)$ addition non-constant-time. In particular, there exists a timing side-channel which can be used to determine whether the inputs points are equal.

In context of the Resp and Update phases of our leakage-resilient ID protocol, we do not consider the special cases when one or both of the input points are the point at infinity. This is because the input points are obtained from ECSM operations at earlier stages of the protocol, and we ensure that the corresponding scalars are smaller than $q$, where $q$ is the curve order.

We refer to the complete addition formula proposed in [126] in order to implement constant-time point addition. A projective coordinate representation $(X : Y : Z)$ is used, such that $x = X/Z$ and $y = Y/Z$, which homogenizes the elliptic curve equation to $E/\mathbb{F}_p: Y^2Z = X^3 + aXZ^2 + bZ^3$. For the BN254 curve, we have $a = 0$ and $b = 2$. Also, it is safe to assume that $Z = 1$ for both input points, that is, consider them to be affine points, since they are not the point at infinity, as explained earlier.

As shown in Table 4.2, point addition using this formula involves 13 modular additions / subtractions, 12 modular multiplications and 1 modular inversion. The modular inversion, along with 2 modular multiplications, is used to convert the output point from projective to affine coordinates. Compared to the traditional point addition formulae, this method is roughly 2x slower on the average, but is constant-
time. Since all $\mathbb{F}_p$ arithmetic is accelerated in hardware and point additions are much faster than ECSM, this slow-down has negligible impact on the overall performance of the protocol.

### 4.4.5 Measurement Results

Fig. 4-4 shows the measurement setup for our test chip. The test chip was placed inside a QFN64 socket mounted on a custom PCB, which was then connected to an Opal Kelly XEM7001 FPGA board. The FPGA was used to emulate the SD card program memory and to control the chip GPIOs. A Keithley 2602A source meter was used to supply power to the chip and measure current consumption. A Python-based interface was used to configure both the FPGA and the source meter from a PC.

![Test Setup](image)

Figure 4-4: Test Setup

Table 4.4 presents performance and energy consumption for different phases of the ID protocol, measured at 0.8 V and 16 MHz. The Commit phase involves 3 ECSMs and 3 pairings, where the final exponentiation is performed only once at the end, as
Table 4.4: Measurement results at 0.8 V and 16 MHz

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cycles</th>
<th>Time</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>$\approx 41$M</td>
<td>2.57 s</td>
<td>1.5 mJ</td>
</tr>
<tr>
<td>Resp</td>
<td>$\approx 1.5$M</td>
<td>94.6 ms</td>
<td>51.8 $\mu$J</td>
</tr>
<tr>
<td>Update</td>
<td>$\approx 3.0$M</td>
<td>187.9 ms</td>
<td>103.0 $\mu$J</td>
</tr>
</tbody>
</table>

explained in section 4.3. It takes around 41M cycles, which translates to 2.57 s runtime. The corresponding energy consumption is 1.5 mJ. The Resp and Update phases take 1.5M and 3.0M cycles respectively, which is much faster than Commit since there are no pairing computations involved. The corresponding energy consumption are also an order of magnitude smaller than Commit. Further performance gains are possible by choosing to pre-compute a batch of key-update and commit messages offline.

The total code size of our protocol implementation is around 15.9 KB, so it just fits in the instruction cache. A detailed breakdown is presented in Figure 4-5. The maximum data memory usage is around 11.5 KB, which includes program read-only data, heap and stack usage.

![Figure 4-5: Code Size Breakdown](image-url)
4.5 Conclusion

This chapter presents an ID scheme that is shown to be leakage resilient under the SXDH assumption. We demonstrate the practicality of this scheme by implementing it over the BN254 Elliptic Curve on an embedded RISCV processor. We show real-time performance with 2.9 s protocol latency and consuming 1.65 mJ of energy. Pre-computing the key-updates allows us to further improve online performance and requires just 95ms and 51.8 μJ.
Chapter 5

Gazelle: Accelerated Secure Neural Network Inference

5.1 Introduction

Fueled by the massive influx of data, sophisticated algorithms and extensive computational resources, modern machine learning has found surprising applications in such diverse domains as medical diagnosis [56,149], facial recognition [139] and credit risk assessment [8]. Often these algorithms operate in a supervised machine learning setting which proceeds in two phases: a training phase where a labeled dataset is turned into a model, and an inference or classification or prediction phase where the model is used to predict the label of a new unlabelled data point. This work in particular targets a class of complex and powerful machine learning models, namely Convolutional Neural Networks (CNN) which are particularly suited to a variety of image classification tasks [69,78,92].

An emerging use-case for machine learning models (including CNNs) is the Prediction-as-a-Service (PaaS) setting. Here the training and model development is out-sourced to a large organization which might possibly use its proprietary data. The organization then wants to monetize the model by deploying a service that allows clients to upload their inputs and receive predictions for a price.

For example, large hospitals that own massive amounts of medical data can de-
velop and monetize models that predict occurrence of diseases given a patient’s medical history and their genetic data. Alternatively financial companies that own data about market trends can build models to predict the movement of stock prices.

These organizations may choose to make the model (in our setting, the architecture and parameters of the CNN) freely available for public consumption. This is undesirable for at least two reasons: first, once the model is given away, there is clearly no opportunity for the organization to monetize it, potentially removing its incentives to undergo the expensive data curating, cleaning and training phases; and secondly, the model, which has been trained on private organizational data, may reveal information about users that contributed to the dataset, violating their privacy and perhaps even regulations such as HIPAA and FERPA.

A second solution is for the organization to build a web service that hosts the model and provides predictions for a small fee. However, this is also undesirable for at least two reasons: first, the users of such a service will rightfully be concerned about the privacy of the inputs they are providing to the web service; and secondly, the organization may not even want to know the user inputs for reasons of legal liability in case of a future data breach.

The goal of our work is to provide practical solutions to this conundrum of secure neural network inference. More concretely, we aim to provide a way for the organization and its users to interact in such a way that the user eventually obtains the prediction (without learning the model) and the organization obtains no information about the user’s input.

Modern cryptography provides us with many tools, in particular fully homomorphic encryption and garbled circuits, that can help us solve this issue. A key take-away from our work is that both techniques have their limitations; understanding their precise trade-offs and using a combination of them judiciously in an application-specific manner helps us overcome the individual limitations and achieve substantial gains in performance. Indeed, several recent works [68,99,110,128,133] have built systems that address the problem of secure neural network inference using these cryptographic tools, and our work improves on all of them. Let us begin by discussing these two
techniques and their relative merits and shortcomings.

5.1.1 Homomorphic Encryption

Fully Homomorphic Encryption (FHE), is an encryption method that allows anyone to compute an arbitrary function \( f \) on an encryption of \( x \), without decrypting it and without knowledge of the private key [28, 66, 130]. Using just the encryption of \( x \) one can obtain an encryption of \( f(x) \). Weaker versions of FHE, collectively called Partially Homomorphic Encryption (PHE), permit the computation of a subset of all functions, typically functions that perform only additions - Additively Homomorphic Encryption (AHE) [118] or functions that can be computed by depth-bounded arithmetic circuits - Leveled Homomorphic Encryption (LHE) [27, 29, 58]. Although LHE schemes allow us to represent more complex functions when compared with AHE schemes, this increased expressive power comes at the cost of a higher computational overhead. Hence it is often desirable to use the simplest homomorphic scheme that will suffice for a given application. Recent efforts, both in theory and in practice have given us large gains in the performance of several types of homomorphic schemes [27, 36, 37, 67, 74, 131] allowing us to implement a larger class of applications with better security guarantees. In particular this has the potential to protect against data leakage from weaker encryption paradigms such as deterministic or order-preserving encryption [4] used in past work [123].

The major bottleneck for these techniques, notwithstanding these recent developments, is their computational complexity. The computational cost of LHE, for example, grows dramatically with the depth of the circuit that the scheme needs to support. Indeed, the recent CryptoNets system gives us a protocol for secure neural network inference using LHE [68]. LHE is best suited to computing low-degree polynomial expressions. On the other hand commonly used non-linear activations such as Rectified Linear Unit (ReLU) and Sigmoid do not have accurate low degree polynomial approximations. As such LHE based schemes like CryptoNets has two major shortcomings. First, they need to change the structure of neural networks and retrain them with special LHE-friendly non-linear activation functions such as
the square function. This has a potentially negative effect on the accuracy of these models. Secondly, and perhaps more importantly, even with these changes, the computational cost is prohibitively large. For example, on a neural network trained on the MNIST dataset, the end-to-end latency of CryptoNets is 297.5 seconds, in stark contrast to the 30 milliseconds end-to-end latency of Gazelle. This problem only worsens as the network at hand increases in complexity. In spite of the use of interaction, our online bandwidth per inference for this network is a mere 0.05MB as opposed to the 372MB required by CryptoNets.

In contrast to the LHE scheme in CryptoNets, Gazelle employs a much simpler Packed Additively Homomorphic Encryption (PAHE) scheme, which we show can support very fast matrix-vector multiplications and convolutions. Lattice-based AHE schemes come with powerful features such as Single Instruction Multiple Data (SIMD) evaluation and automorphisms (described in detail in Section 5.3) which make them the ideal tools for common linear-algebraic computations. The second key take-away from our work is that even in applications where only additive homomorphisms are required, lattice-based AHE schemes far outperform other AHE schemes such as the Paillier scheme both in computational and communication complexity.

5.1.2 Secret Sharing and Garbled Circuits

Yao’s garbled circuits [158] and the secret-sharing based Goldreich-Micali-Wigderson (GMW) protocol [70] are two leading methods for the task of secure two-party computation (2PC). After three decades of theoretical and applied work improving and optimizing these protocols, we now have very efficient implementations, e.g., [41, 42, 53, 129]. The modern versions of these techniques have the advantage of being computationally inexpensive, partly because they rely on symmetric-key cryptographic primitives such as Advanced Encryption Standard (AES) and SHA and use them in a clever way [21], because of hardware support in the form of the Intel AES-NI instruction set, and because of techniques such as oblivious transfer extension [21, 83] which limit the use of public-key cryptography to an offline reusable pre-processing phase.
The major bottleneck for these techniques is their communication complexity. Indeed, three recent works followed the garbled circuits paradigm and designed systems for secure neural network inference: the SecureML system [110], the MiniONN system [99], the DeepSecure system [133].

DeepSecure uses garbled circuits alone; SecureML uses Paillier’s AHE scheme to speed up some operations; and MiniONN uses a weak form of lattice-based AHE to generate “multiplication triples” similar to the SPDZ multiparty computation framework [41]. Our key claim is that understanding the precise trade-off point between AHE and garbled circuit-type techniques allows us to make optimal use of both and achieve large net computational and communication gains. In particular, in Gazelle, we use optimized AHE schemes in a completely different way from MiniONN: while they employ AHE as a pre-processing tool for generating triples, we use AHE to dramatically speed up linear algebra directly.

For example, on a neural network trained on the CIFAR-10 dataset, the most efficient of these three protocols, namely MiniONN, has an online bandwidth cost of 6.2GB whereas Gazelle has an online bandwidth cost of 0.3GB. In fact, we observe across the board a reduction of 20-80× in the online bandwidth per inference which gets better as the networks grow in size. In the Local Area Network (LAN) setting, this translates to an end-to-end latency of 3.6s versus the 72s for MiniONN.

Even when comparing to systems such as Chameleon [128] that rely on trusted third-party dealers, we observe a 30× reduction in online run-time and 2.5× reduction in online bandwidth, while simultaneously providing a pure two-party solution, without relying on third-party dealers. (For more detailed performance comparisons with all these systems, we refer the reader to Section 5.10).

5.1.3 (F)HE or Garbled Circuits?

To use (Fully) Homomorphic Encryption ((F)HE) and garbled circuits optimally, we need to understand the precise computational and communication trade-offs between them. Additionally, we need to (a) identify applications and the right algorithms for these applications; (b) partition these algorithms into computational sub-routines
where each of these techniques outperforms the other; and (c) piece together the right solutions for each of the sub-routines in a seamless way to get a secure computation protocol for the entire application. Let us start by recapping the trade-offs between (F)HE and garbled circuits.

Roughly speaking, homomorphic encryption performs better than garbled circuits when (a) the computation has small multiplicative depth, ideally multiplicative depth 0 meaning that we are computing a linear function; and (b) the Boolean circuit that performs the computation has large size, say quadratic in the input size. Matrix-vector multiplication (namely, the operation of multiplying a plaintext matrix with an encrypted vector) provides us with exactly such a scenario. Furthermore, the most time-consuming computations in a convolutional neural network are indeed the convolutional layers (which are nothing but a special type of matrix-vector multiplication). The key-idea is that although these operations require many multiplications the degree for both fully-connected and convolutional layers is just one. Thus one can effectively use SIMD multiplication schemes for these layers. The non-linear computations in a CNN such as the ReLU or MaxPool functions can be written as simple linear-size boolean circuits which are best computed using garbled circuits. This analysis is the guiding philosophy that enables the design of Gazelle. For detailed descriptions of CNNs, we refer the reader to Section 5.2.

5.1.4 Our System: Gazelle

The main contribution of this work is Gazelle, a framework for secure evaluation of CNNs. It consists of three components:

- The first component is the Gazelle Homomorphic Layer which consists of very fast implementations of three basic homomorphic operations: SIMD addition, SIMD scalar multiplication, and automorphisms (For a detailed description of these operations, see Section 5.3). Our innovations in this part consist of techniques for division-free arithmetic and techniques for lazy modular reductions. In fact, our implementation of the first two of these homomorphic operations
incurs only 10-20x slower than the corresponding operations on plaintext, *when counting the number of clock cycles*.

- The second component is the *Gazelle Linear Algebra kernels* which consists of very fast algorithms for homomorphic matrix-vector multiplications and homomorphic convolutions, accompanied by matching implementations. In terms of the basic homomorphic operations, SIMD additions and multiplications turn out to be relatively cheap whereas automorphisms are very expensive. At a very high level, our innovations in this part consists of several new algorithms for homomorphic matrix-vector multiplication and convolutions that minimize the expensive automorphism operations.

- The third and final component is *Gazelle Network Inference* which uses a judicious combination of garbled circuits together with our linear algebra kernels to construct a protocol for secure neural network inference. Our innovations in this part consist of efficient protocols that switch between secret-sharing and homomorphic representations of the intermediate results and a novel protocol to ensure circuit privacy.

Our protocol also hides strictly more information about the neural network than other recent works such as the MiniONN protocol. We refer the reader to Section 5.2 for more details.

### 5.2 Secure Neural Network Inference

The goal of this section is to describe a clean abstraction of CNNs and set up the secure neural inference problem that we will tackle in the rest of the chapter. A CNN takes an input and processes it through a sequence of *linear* and *non-linear* layers in order to classify it into one of the potential classes. An example CNN is shown in Figure 5-1.
5.2.1 Linear Layers

The linear layers, shown in Figure 5-1 in red, can be of two types: convolutional (Conv) layers or fully-connected (FC) layers.

Conv Layers. We represent the input to a Conv layer by the tuple \((w_i, h_i, c_i)\) where \(w_i\) is the image width, \(h_i\) is the image height, and \(c_i\) is the number of input channels. In other words, the input consists of \(c_i\) many \(w_i \times h_i\) images. The convolutional layer is then parameterized by \(c_o\) filter banks each consisting of \(c_i\) many \(f_w \times f_h\) filters. This is represented in short by the tuple \((f_w, f_h, c_i, c_o)\). The computation in a Conv layer can be better understood in term of simpler Single-input Single-output (SISO) convolutions. Every pixel in the output of a SISO convolution is computed by stepping a single \(f_w \times f_h\) filter across the input image as shown in Figure 5-2. The output of the full Conv layer can then be parameterized by the tuple \((w_o, h_o, c_o)\) which represents \(c_o\) many \(w_o \times h_o\) output images. Each of these images is associated to a unique filter bank and is computed by the following two-step process shown in Figure 5-2: (i) For each of the \(c_i\) filters in the associated filter bank, compute a SISO convolution with the corresponding channel in the input image, resulting in \(c_i\) many intermediate images; and (ii) summing up all these \(c_i\) intermediate images.

There are two commonly used padding schemes when performing convolutions. In the "valid" scheme, no input padding is used, resulting in an output image that is smaller than the initial input. In particular we have \(w_o = w_i - f_w + 1\) and \(h_o =\)
Figure 5-2: SISO convolutions and multi-channel Conv layers

In the "same" scheme, the input is zero padded such that output image size is the same as the input.

In practice, the Conv layers sometimes also specify an additional pair of stride parameters \((s_w, s_h)\) which denotes the granularity at which the filter is stepped. After accounting for the strides, the output image size \((w_o, h_o)\), is given by \(\left\lfloor (w_i - f_w + 1)/s_w \right\rfloor, \left\lfloor (h_i - f_h + 1)/s_h \right\rfloor\) for valid style convolutions and \(\left\lfloor w_i/s_w \right\rfloor, \left\lfloor h_i/s_h \right\rfloor\) for same style convolutions.

**FC Layers.** The input to a FC layer is a vector \(v_i\) of length \(n_i\) and its output is a vector \(v_o\) of length \(n_o\). A fully connected layer is specified by the tuple \((W, b)\) where \(W\) is \((n_o \times n_i)\) weight matrix and \(b\) is an \(n_o\) element bias vector. The output
is specified by the following transformation: \( v_o = Wv_i + b \).

The key observation that we wish to make is that the number of multiplications in the Conv and FC layers are given by \((w_o \cdot h_o \cdot c_o \cdot f_w \cdot f_h \cdot c_i)\) and \(n_i \cdot n_o\), respectively. This makes both the Conv and FC layer computations quadratic in the input size. This fact guides us to use homomorphic encryption rather than garbled circuit-based techniques to compute the convolution and fully connected layers, and indeed, this insight is at the heart of the much of the speedup achieved by Gazelle.

5.2.2 Non-Linear Layers

The non-linear layers, shown in Figure 5-1 in blue, consist of an activation function that acts on each element of the input separately or a pooling function that reduces the output size. Typical non-linear functions can be one of several types: the most common in the convolutional setting are max-pooling functions and ReLU functions.

The key observation that we wish to make in this context is that all these functions can be implemented by circuits that have size linear in the input size. For example consider the ReLU function, which can be represented as follows \( \text{ReLU}(x) = \max(0, x) \). Implementing this function requires a single comparison operation and hence the number of gates in the ReLU boolean circuit is linear in the bit-size of \( x \). Thus evaluating these functions using conventional 2PC approaches does not impose any additional asymptotic communication penalty when compared with just communicating the inputs.

For more details on CNNs, we refer the reader to [144].

5.2.3 Secure Inference: Problem Description

In our setting, there are two parties \( A \) and \( B \) where \( A \) holds a CNN and \( B \) holds an input to the network, typically an image. We make the distinction between the structure of the CNN which includes the number of layers, the size of each layer, and the activation functions applied in layer, versus the parameters of the CNN which includes all the weights and biases that describe the convolution and the fully
connected layers.

We wish to design a protocol that \( A \) and \( B \) engage in at the end of which \( B \) obtains the classification result (and potentially the network structure), namely the output of the final layer of the neural network, whereas \( A \) obtains nothing.

The Threat Model

Our threat model is the same as in previous works, namely the SecureML, MiniONN and DeepSecure systems and our techniques, as we argue below, leak even less information than in these works.

To be more precise, we consider semi-honest corruptions as in [99, 110, 133], i.e., \( A \) and \( B \) adhere to the software that describes the protocol, but attempt to infer information about the other party’s input (the network parameters or the image, respectively) from the protocol transcript. We ask for the cryptographic standard of ideal/real security [70, 71]. Two comments are in order about this ideal functionality.

The first is an issue specific to the ideal functionality instantiated in this and past work, i.e., the ideal functionality does not completely hide the network structure. We argue, however, that it does hide the important aspects which are likely to be proprietary. In particular, the ideal functionality and our realization hides all the weights and biases in the convolution and the fully connected layers. Secondly, we also hide the filter and stride size in the convolution layers, as well as information as to which layers are convolutional layers and which are fully connected. We do reveal the number of layers and the size\(^1\) (the number of hidden nodes) of each layer. As for party \( B \)’s security, we hide the entire image, but not its size, from party \( A \).

In contrast, other protocols for secure neural network inference such as the MiniONN protocol [99] reveal strictly more information, e.g., they reveal the filter size. This knowledge coupled with the number of layers would allow the adversary to learn the structure of the network. In the case where the training data was publicly available one could then potentially train a copy of the original network.

\(^1\)One can potentially hide this information by padding the network with dummy operations at a proportional computational expense
A second, more subtle, issue is with the definition of the ideal functionality which implements secure network inference. Since such functionality, must at a bare minimum, give $B$ access to the classification output, $B$ maybe be able to train a new classifier to mimic these classification results. This attack is called model stealing [147]. Note that model stealing with limited queries is essentially equivalent to a supervised learning task with access to a limited training dataset. Thus a potential model stealing adversary could train such classifier without access to $B$ by simply asking a domain expert to classify his limited set of test-images. One potential solution is to limit the number of classification queries that $A$ is allowed to make of the model. This can be a practical solution in a try-before-buy scenario where $B$ only needs access to limited set of classifications to test the performance of the network before purchasing the network parameters from $A$. We remark that designing (potentially-noisy) classifiers which are intrinsically resilient to model stealing is an interesting open machine learning problem.

Chapter Organization

The rest of the chapter is organized as follows. We first describe our abstraction of a PAHE that we use through the rest of the chapter. We then provide an overview of the entire Gazelle protocol in section 5.5. In the next two sections, Section 5.6 and 5.8, we elucidate the most important technical contributions of the chapter, namely the *Gazelle Linear Algebra Kernels* for fast matrix-vector multiplication and convolution. We then present detailed benchmarks on the implementation of the *Gazelle Homomorphic Layer* and the linear algebra kernels in Section 5.9. Finally, we describe the evaluation of neural networks such as ones trained on the MNIST or CIFAR-10 datasets and compare Gazelle’s performance to prior work in Section 5.10.

5.3 Packed Additively Homomorphic Encryption

In this section, we describe a clean abstraction of PAHE schemes that we will use through the rest of the chapter. As suggested by the name, the abstraction will sup-
port packing multiple plaintexts into a single ciphertext, performing SIMD homomorphic additions (SIMDAdd) and scalar multiplications (SIMDScMult), and permuting the plaintext slots (Perm). In particular, we will never need or use homomorphic multiplication of two ciphertexts. This abstraction can be instantiated with essentially all modern lattice-based homomorphic encryption schemes, e.g., [27, 29, 58, 67].

For the purposes of this chapter, a private-key PAHE suffices. In such an encryption scheme, we have a (randomized) encryption algorithm (PAHE.Enc) that takes a plaintext message vector \( u \) from some message space and encrypts it using a key \( sk \) into a ciphertext denoted as \([u]\), and a (deterministic) decryption algorithm (PAHE.Dec) that takes the ciphertext \([u]\) and the key \( sk \) and recovers the message \( u \). Finally, we also have a homomorphic evaluation algorithm (PAHE.Eval) that takes as input one or more ciphertexts that encrypt messages \( M_0, M_1, \ldots \), and outputs another ciphertext that encrypts a message \( M = f(M_0, M_1, \ldots) \) for some function \( f \) constructed using the SIMDAdd, SIMDScMult and Perm operations. We require Indistinguishability under Chosen Plaintext Attack (IND-CPA) Security, which requires that ciphertexts of any two messages \( u \) and \( u' \) are computationally indistinguishable.

The lattice-based PAHE constructions that we consider in this chapter are parameterized by four constants: (1) the cyclotomic order \( m \), (2) the ciphertext modulus \( q \), (3) the plaintext modulus \( p \) and (4) the standard deviation \( \sigma \) of a symmetric discrete Gaussian noise distribution \( \chi \).

The number of slots in a packed PAHE ciphertext is given by \( n = \phi(m) \) where \( \phi \) is the Euler Totient function. Thus, plaintexts can be viewed as length-\( n \) vectors over \( \mathbb{Z}_p \) and ciphertexts are viewed as length-\( n \) vectors over \( \mathbb{Z}_q \). All fresh ciphertexts start with an inherent noise \( \eta \) sampled from the noise distribution \( \chi \). As homomorphic computations are performed \( \eta \) grows continually. Correctness of PAHE.Dec is predicated on the fact that \( |\eta| < q/(2p) \), thus setting an upper bound on the complexity of the possible computations.

We note that although the exact value of noise in the final output depends on the plaintext we are computing on, one can bound the noise for each of the individual homomorphic operations. One can then compose these bounds to generate a bound
for the noise in the final output. By validating this bound against the correctness condition for \texttt{PAHE.Dec}, one can ensure that the final noise does result in incorrect outputs.

In order to guarantee security we require a minimum value of $\sigma$ (based on $q$ and $n$), $q \equiv 1 \pmod{m}$ and $p$ is co-prime to $q$. Additionally, in order to minimize noise growth in the homomorphic operations we require that the magnitude of $r \equiv q \pmod{p}$ be as small as possible. This when combined with the security constraint results in an optimal value of $r = \pm 1$.

In the sequel, we describe in detail the three basic operations supported by the homomorphic encryption schemes together with their associated asymptotic cost in terms of (a) the run-time, and (b) the noise growth. Later, in Section 5.9, we will provide concrete micro-benchmarks for each of these operations implemented in the \textsc{gazelle} library.

5.3.1 Addition: \texttt{SIMDAdd}

Given ciphertexts $[u]$ and $[v]$, \texttt{SIMDAdd} outputs an encryption of their componentwise sum, namely $[u + v]$.

The asymptotic run-time for homomorphic addition is $n \cdot \text{CostAdd}(q)$, where \text{CostAdd}(q) is the run-time for adding two numbers in $\mathbb{Z}_q = \{0, 1, \ldots, q - 1\}$. The noise growth is at most $\eta_u + \eta_v$ where $\eta_u$ (resp. $\eta_v$) is the amount of noise in $[u]$ (resp. in $[v]$).

5.3.2 Scalar Multiplication: \texttt{SIMDScMult}

If the plaintext modulus is chosen such that $p \equiv 1 \pmod{m}$, we can also support a SIMD componentwise product. Thus given a ciphertext $[u]$ and a plaintext $v$, we can output an encryption $[u \circ v]$ (where $\circ$ denotes component-wise multiplication of vectors).

The asymptotic run-time for homomorphic scalar multiplication is $n \cdot \text{CostMult}(q)$, where $\text{CostMult}(q)$ is the run-time for multiplying two numbers in $\mathbb{Z}_q$. The noise
growth is at most $\eta_{\text{mult}} \cdot \eta_u$ where $\eta_{\text{mult}} \approx ||v||_\infty \cdot \sqrt{n}$ is the multiplicative noise growth of the SIMD scalar multiplication operation.

For a reader familiar with homomorphic encryption schemes, we note that $||v||_\infty$ is the largest value in the coefficient representation of the packed plaintext vector $v$, and thus, even a binary plaintext vector can result in $\eta_{\text{mult}}$ as high as $p \cdot \sqrt{n}$. In practice, we alleviate this large multiplicative noise growth by bit-decomposing the coefficient representation of $v$ into $\log(p/2^{w_{\text{pt}}})$ many $w_{\text{pt}}$-sized chunks $v_k$ such that $v = \sum 2^{w_{\text{pt}} \cdot k} \cdot v_k$. We refer to $w_{\text{pt}}$ as the plaintext window size.

We can now represent the product $[u \circ v]$ as $\sum [u_k \circ v_k]$ where $u_k = [2^{w_{\text{pt}} \cdot k} \cdot u]$. Since $||v_k||_\infty \leq 2^{w_{\text{pt}}}$ the total noise in the multiplication is bounded by $\sum_k 2^{w_{\text{pt}}} \cdot \sqrt{n} \cdot \eta_{u_k}$ as opposed to $p \cdot \sqrt{n} \cdot \eta_u$. The only caveat is that we need access to low noise encryptions $[u_k]$ as opposed to just $[u]$ as in the direct approach.

### 5.3.3 Slot Permutation: Perm

Given a ciphertext $[u]$ and one of a set of primitive permutations $\pi$ defined by the scheme, the Perm operation outputs a ciphertext $[u_\pi]$, where $u_\pi$ is defined as $(u_{\pi(1)}, u_{\pi(2)}, \ldots, u_{\pi(n)})$, namely the vector $u$ whose slots are permuted according to the permutation $\pi$. The set of permutations that can be supported depends on the structure of the multiplicative group $\mod m$ i.e. $(\mathbb{Z}/m\mathbb{Z})^\times$. When $m$ is prime, we have $n = (m - 1)$ slots and the permutation group supports all cyclic rotations of the slots, i.e. it is isomorphic to $C_n$ (the cyclic group of order $n$). When $m$ is a sufficiently large power of two ($m = 2^k, m \geq 8$), we have $n = 2^{k-1}$ and the set of permutations is isomorphic to the set of half-rotations i.e. $C_{n/2} \times C_2$, as illustrated in Figure 5-4.

Permutations are by far the most expensive operations in a homomorphic encryption scheme. At a high-level the PAHE ciphertext vectors represent polynomials. The permutation operation requires transforming these polynomials from evaluation to coefficient representations and back. These transformations can be efficiently computed using the number theoretic transform (NTT) and its inverse, both of which are finite-field analogues of their real valued Discrete Fourier Transform counterparts. Both the NTT and $\text{NTT}^{-1}$ have an asymptotic cost of $\Theta(n \log n)$. As seen in [28],
Figure 5-3: Ciphertext Structure and Operations. Here, \( n \) is the number of slots, \( q \) is the size of ciphertext space (so a ciphertext required \( \lfloor \log_2 q \rfloor \) bits to represent), \( p \) is the size of the plaintext space (so a plaintext can have at most \( \lfloor \log_2 p \rfloor \) bits), and \( \eta \) is the amount of noise in the ciphertext.

Figure 5-4: A Plaintext Permutation in action. The permutation \( \pi \) in this example swaps the first and the second slots, and also the third and fourth slots. The operation incurs a noise growth from \( \eta \) to \( \eta' \approx \eta + \eta_{\text{rot}} \). Here, \( \eta_{\text{rot}} \approx n \log q \cdot \eta_0 \) where \( \eta_0 \) is some small "base noise".

we need to perform \( \Theta(\log q) \) \( \text{NTT}^{-1} \) to control \( \text{Perm} \) noise growth. The total cost of \( \text{Perm} \) is therefore \( \Theta(n \log n \log q) \) operations. The noise growth is additive, namely, \( \eta_{\text{ux}} = \eta_{\text{ux}} + \eta_{\text{rot}} \) where \( \eta_{\text{rot}} \) is the additive noise growth of a permutation operation.

Detailed analysis of how to select an efficient set of \( \text{PAHE} \) parameters is presented in Section 5.4.

### 5.3.4 Paillier vs. Lattice-based PAHE

The \( \text{PAHE} \) scheme used in Gazelle is dramatically more efficient than conventional Paillier based \( \text{AHE} \). Homomorphic addition of two Paillier ciphertexts corresponds to a modular multiplication modulo a large Rivest-Shamir-Adleman (RSA)-like modulus (3072bits) as opposed to a simple addition \( \mod q \) as seen in \( \text{SIMDAdd} \). Similarly multiplication by a plaintext turns into a modular exponentiation for Paillier. Furthermore the large sizes of the Paillier ciphertexts makes encryption of single small integers extremely bandwidth-inefficient. In contrast, the notion of packing provided
by lattice-based schemes provides us with a SIMD way of packing many integers into one ciphertext, as well as SIMD evaluation algorithms. We are aware of one system [136] that tries to use Paillier in a SIMD fashion; however, this lacks two crucial components of lattice-based AHE, namely the facility to multiply each slot with a separate scalar, and the facility to permute the slots. We are also aware of a method of mitigating the first of these shortcomings [82], but not the second. Our fast homomorphic implementation of linear algebra uses both these features of lattice-based AHE, making Paillier an inherently unsuitable substitute.

5.4 Parameter Selection for PAHE

Parameter selection for PAHE requires a delicate balance between the homomorphic evaluation capabilities and the target security level. We detail our procedure for parameter selection to meet a target security level of 128 bits. We first set our plaintext modulus to be 20 bits to represent the fixed point inputs (the bit-length of each pixel in an image) and partial sums generated during the neural network evaluation. Next, we require that the ciphertext modulus be close to, but less than, 64 bits in order to ensure that each ciphertext slot fits in a single machine word while maximizing the potential noise margin available during homomorphic computation.

The Perm operation in particular presents an interesting tradeoff between the simplicity of possible rotations and the computational efficiency of the Number Theoretic Transform (NTT). A prime $m$ results in a (simpler) cyclic permutation group but necessitates the use of an expensive Bluestein transform. Conversely, the use of $m = 2^k$ allows for a \(8\times\) more efficient Cooley-Tukey style NTT at the cost of an awkward permutation group that only allows half-rotations. In this work, we opt for the latter and adapt our linear algebra kernels to deal with the structure of the permutation group. Based on the analysis of [71], we set $m = 4096$ and $\sigma = 4$ to obtain our desired security level.

Our chosen bit-width for $q$, namely 60 bits, allows for lazy reduction, i.e. multiple additions may be performed without overflowing a machine word before a reduc-
tion is necessary. Additionally, even when $q$ is close to the machine word-size, we can replace modular reduction with a simple sequence of addition, subtraction and multiplications. This is done by choosing $q$ to be a pseudo-Mersenne number.

Next, we detail a technique to generate prime moduli that satisfy the above correctness and efficiency properties, namely:

1. $q \equiv 1 \pmod{m}$
2. $p \equiv 1 \pmod{m}$
3. $|q \mod p| = |r| \approx 1$
4. $q$ is pseudo-Mersenne, i.e. $q = 2^{60} - \delta, (\delta < \sqrt{q})$

Below, we describe a fast method to generate $p$ and $q$ (We remark that the obvious way to do this requires at least $p \approx 2^{20}$ primality tests, even to satisfy the first three conditions).

Since we have chosen $m$ to be a power of two, we observe that $\delta \equiv -1 \pmod{m}$. Moreover, $r \equiv q \pmod{p}$ implies that $\delta \equiv (q - r) \pmod{p}$. These two Chinese Remainder Theorem (CRT) expressions for $\delta$ imply that given a prime $p$ and residue $r$, there exists a unique minimal value of $\delta \pmod{(p \cdot m)}$.

Based on this insight our prime selection procedure can be broken down into three steps:

1. Sample for $p \equiv 1 \pmod{m}$ and sieve the prime candidates.
2. For each candidate $p$, compute the potential $2|r|$ candidates for $\delta$ (and thus $q$).
3. If $q$ is prime and $\delta$ is sufficiently small accept the pair $(p, q)$.

Heuristically, this procedure needs $\log(q)(p \cdot m)/(2|r|\sqrt{q})$ candidate primes $p$ to sieve out a suitable $q$. Since $p \approx 2^{20}$ and $q \approx 2^{64}$ in our setting, this procedure is very fast. A list of reduction-friendly primes generated by this approach is tabulated in Table 5.1. Finally note that when $[\log(p)] \cdot 3 < 64$ we can use Barrett reduction to speed-up reduction mod $p$. 

98
Table 5.1: Prime Selection for PAHE

| \[ \log(p) \] | \( p \) | \( q \) | \( |r| \) |
|----------------|--------|--------|--------|
| 18             | 307201 | \( 2^{60} - 2^{12} \cdot 63549 + 1 \) | 1      |
| 22             | 5324801 | \( 2^{60} - 2^{12} \cdot 122130 + 1 \) | 1      |
| 26             | 115351553 | \( 2^{60} - 2^{12} \cdot 9259 + 1 \) | 1      |
| 30             | 1316638721 | \( 2^{60} - 2^{12} \cdot 54778 + 1 \) | 2      |

The impact of the selection of reduction-friendly primes on the performance of the PAHE scheme is described in section 5.9.

5.5 Our Protocol at a High Level

Our protocol evaluates the neural network under consideration is based on the alternating use of PAHE and Garbled Circuits (GC) as shown in Figure 5-5. We will next explain the flow of the protocol and the explain how one can efficiently and securely convert between the data representations required for the cryptographic primitives.

The main invariant that the protocol maintains is that at the start of the PAHE phase the server and the client posses an additive share \( c_y, s_y \) of the client’s input \( y \). At the very beginning of the computation this can be accomplished by the trivial share \( (c_y, s_y) = (y, 0) \).

In order to evaluate a linear layer, we start with the client \( B \) first encrypting their share using the PAHE scheme and sending it to the server \( A \). \( A \) in turn homomorphically adds her share \( s_y \) to obtain an encryption of \( c_y + s_y = y \). The security of the homomorphic encryption scheme guarantees that \( B \) cannot recover \( y \) from this encryption. The server \( A \) then uses the Gazelle homomorphic neural network kernel for the linear layer (which is either convolution or fully connected). The result is a packed ciphertext that contains the input to the first non-linear (ReLU) layer. The homomorphic scheme ensures that the \( A \) learns nothing about \( B \)'s input. \( B \) has not received any input from \( A \) yet and thus has no way of learning the model parameters.

In preparation for the evaluation of the subsequent non-linear activation layer \( A \) must transform her PAHE ciphertext into additive shares. At the start of this step...
A holds a ciphertext $[x]$ (where $x$ is a vector) and $B$ holds the private key. The first step is to transform this ciphertext such that both $A$ and $B$ hold an additive secret sharing of $x$. This is accomplished by the server $A$ adding a random vector $r$ to her ciphertext homomorphically to obtain an encryption $[x + r]$ and sending it to the client $B$. The client $B$ then decrypts this message to get his share. Thus the server $A$ sets her share $s_x = r$ and $B$ sets his share $c_x = x + r \pmod{p}$. Since the $A$ chooses $r$ uniformly at random $s_x$ does not contain any information about either the model or the $B$’s input. Since $B$ does not know $r$, $c_x$ has a uniform random distribution from $B$’s perspective. Moreover the security of the PAHE scheme ensures that $A$ has no way of figuring out what $c_x$ is.

We next evaluate the non-linear activation using Yao’s GC protocol. At the start of this step both parties posses additive shares $(c_x, s_x)$ of the secret value of $x$ and want to compute $y = \text{ReLU}(x)$ without revealing it completely to either party. We
evaluate the non-linear activation function ReLU (in parallel for each component of $x$) to get a secret sharing of the output $y = \text{ReLU}(x)$. This is done using our circuit from Figure 5-6, described in more detail below. The output of the garbled circuit evaluation is a pair of shares $s_y$ (for the server) and $c_y$ (for the client) such that $s_y + c_y = y \pmod{p}$. The security argument is exactly the same as after the first step, i.e. neither party has complete information and both shares appear uniformly random to their respective owners.

Once this is done, we are back where we started. The next linear layer (either fully connected or convolutional) is again evaluated using the Gazelle homomorphic neural network kernel, followed by Yao’s garbled circuit protocol for the next non-linear layer, so we rinse and repeat until we evaluate the full network. We make the following two observations about our proposed protocol:

1. By using AHE for the linear layers, we ensure that the communication complexity of protocol is linear in the number of layers and the size of inputs for each layer.

2. At the end of the garbled circuit protocol we have an additive share that can be encrypted afresh. As such, we can view the re-encryption as an interactive bootstrapping procedure that clears the noise introduced by any previous homomorphic operation.

For the second step of the outline above, we employ the Boolean circuit described
in Figure 5-6. The circuit takes as input three vectors: \( s_x = r \) and \( s_y = r' \) (chosen at random) from the server, and \( c_x \) from the client. The first block of the circuit computes the arithmetic sum of \( s_x \) and \( c_x \) over the integers and subtracts \( p \) from to obtain the result mod \( p \). (The decision of whether to subtract \( p \) or not is made by the multiplexer). The second block of the circuit computes a ReLU function. The third block adds the result to \( s_y \) to obtain the client's share of \( y \), namely \( c_y \). For more detailed benchmarks on the ReLU and MaxPool garbled circuit implementations, we refer the reader to Section 5.10.

In our evaluations, we consider ReLU, MaxPool and the square activation functions, the first two are by far the most commonly used ones in convolutional neural network design [77,92,140,145]. Note that the square activation function popularized for secure neural network evaluation in [68] can be efficiently implemented by a simple interactive protocol that use the PAHE scheme to generate the cross-terms.

The use of an IND-CPA secure PAHE scheme for evaluating the linear layers guarantees the privacy of the client's inputs. However the PAHE scheme must also guarantee the confidentiality of the server's input, in other words, it should be circuit-private. Prior work addresses this problem in two ways. The first approach called noise-flooding adds a large amount of noise to the final ciphertext [66] to obscure any information leaked through the ciphertext noise. The second technique relies on bootstrapping, either using garbled circuits [65] or using the full power of an FHE scheme [50]. At a high-level both these approaches aim to securely decrypt and re-encrypt a given ciphertext in a way that the final noise is independent of the noise in the initial ciphertext. Noise-flooding causes an undesirable blow-up in the parameters of the underlying PAHE scheme, while the FHE-bootstrapping based solution is well beyond the scope of the simple PAHE schemes we employ. Thus, our solution builds a low-overhead circuit-private interactive decryption protocol to improve the concrete efficiency of the garbled circuit approach (as in [65]) as applied to the Brakerski-Fan-Vercauteren (BFV) scheme [29,58].
5.5.1 Circuit Privacy

At a high level BFV ciphertexts look like a tuple of ring elements \((a, b)\) where \(a\) is chosen uniformly at random and \(b\) encapsulates the plaintext and the ciphertext noise. Both \(a\) and the ciphertext noise are modified in circuit dependent fashion during the process of homomorphic computation and thus may violate circuit privacy. We address the former by simply adding a fresh public-key encryption of zero to the ciphertext to re-randomize \(a\). Information leakage through the noise is handled through interactive decryption. The BFV decryption circuit is given by \(\lceil (a \cdot s + b) / \Delta \rceil\) where \(s\) is the secret key and \(\Delta = \lceil (p/q) \rceil\). Our approach splits the interactive computation of this circuit into 2 phases. First we send the re-randomized \(a\) back to the client who multiplies it with \(s\) to \(a \cdot s\). We then use a garbled circuit to add this to \(b\). We leverage the fact that \(\Delta\) is public to avoid an expensive division inside the garbled circuit. In particular both parties can compute the quotients and remainders modulo \(\Delta\) of their respective inputs and then interactively evaluate a garbled circuit whose size is \(\Omega(n \cdot q)\). Note that in contrast the naive decryption circuit is \(\Omega(n^2 \cdot q)\) sized even without accounting for the division by \(\Delta\).

5.6 Fast Homomorphic Matrix-Vector Multiplication

We next describe the Gazelle homomorphic linear algebra kernels that compute matrix-vector products (for FC layers) and 2-d convolutions (for Conv layers). In this section, we focus on matrix-vector product kernels which multiply a plaintext matrix with an encrypted vector. We start with the easiest to explain (but the slowest and most communication-inefficient) methods and move on to describing optimizations that make matrix-vector multiplication much faster. In particular, our hybrid method (see Table 5.2 and the description below) gives us the best performance among all our homomorphic matrix-vector multiplication methods. For example, multiplying a \(128 \times 1024\) matrix with a length-1024 vector using our hybrid scheme takes about 16ms on a commodity machine. (For detailed benchmarks, we refer the reader to Section 5.9.3). In all the subsequent examples, we will use an FC layer with
### Table 5.2: Comparing matrix-vector product algorithms by operation count, noise growth and number of output ciphertexts

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Perm (Hoisted)</th>
<th>Perm</th>
<th>SIMDScMult</th>
<th>SIMMDAdd</th>
<th>Noise</th>
<th>#out_ct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve</td>
<td>0</td>
<td>$n_o \cdot \log n_i$</td>
<td>$n_o \cdot \log n_i$</td>
<td>$n_o \cdot \log n_i$</td>
<td>$\eta_{naive} := \eta_p \cdot \eta_{mult} \cdot n_i + \eta_{out} \cdot (n_i - 1)$</td>
<td>$n_o$</td>
</tr>
<tr>
<td>Naïve (Output packed)</td>
<td>0</td>
<td>$n_o \cdot \log n_i + n_i - 1$</td>
<td>$2 \cdot n_o \cdot \log n_i + n_i$</td>
<td>$n_o \cdot \log n_i + n_i$</td>
<td>$\eta_{naive} \cdot \eta_{mult} \cdot n_i + \eta_{out} \cdot (n_i - 1)$</td>
<td>1</td>
</tr>
<tr>
<td>Naïve (Input packed)</td>
<td>0</td>
<td>$\frac{n_o n_i}{n} \cdot \log n_i$</td>
<td>$\frac{n_o n_i}{n} \cdot \log n_i$</td>
<td>$\frac{n_o n_i}{n} \cdot \log n_i$</td>
<td>$\eta_p \cdot \eta_{mult} \cdot n_i + \eta_{out} \cdot (n_i - 1)$</td>
<td>$\frac{n_o n_i}{n}$</td>
</tr>
<tr>
<td>Diagonal</td>
<td>$n_i - 1$</td>
<td>0</td>
<td>$n_i$</td>
<td>$n_i$</td>
<td>$\eta_p + \eta_{out}$ \cdot \eta_{mult} \cdot n_i</td>
<td>1</td>
</tr>
<tr>
<td>Hybrid</td>
<td>$\frac{n_o n_i}{n} - 1$</td>
<td>$\log \frac{d_i}{n_o}$</td>
<td>$\frac{n_o n_i}{n} \cdot \log \frac{d_i}{n_o}$</td>
<td>$(\eta_p + \eta_{out}) \cdot \eta_{mult} \cdot n_i + \eta_{out} \cdot (\frac{d_i}{n_o} - 1)$</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

* Rotations of the input with a common PermDecomp.
* Number of output ciphertexts
* All logarithms are to base 2

$n_i$ inputs and $n_o$ outputs as a running example. For simplicity of presentation, unless stated otherwise we assume that $n$, $n_i$ and $n_o$ are powers of two. Similarly we assume that $n_o$ and $n_i$ are smaller than $n$. If not, we can split the original matrix into $n \times n$ sized blocks that are processed independently.

#### The Naïve Method.

In the naïve method, each row of the $n_o \times n_i$ plaintext weight matrix $W$ is encoded into a separate plaintext vectors (see Figure 5-7). Each such vector is of length $n$; where the first $n_i$ entries contain the corresponding row of the matrix and the other entries are padded with 0. These plaintext vectors are denoted $w_0, w_1, \ldots, w_{(n_o - 1)}$. We then use SIMDScMult to compute the componentwise product of with the encrypted input vector $[v]$ to get $[u] = [w_{i} \circ v]$. In order to compute the inner-product what we need is actually the sum of the entries in each of these vectors $u_{i}$.

This can be achieved by a “rotate-and-sum” algorithm, where we first rotate the entries of $[u_{i}]$ by $n_i/2$ positions. The result is a ciphertext whose first $n_i/2$ entries contain the sum of the first and second halves of $u_{i}$. One can then repeat this process for $\log_2 n_i$ iterations, rotating by half the previous rotation on each iteration, to get a ciphertext whose first slot contains the first component of $Wv$. By repeating this procedure for each of the $n_o$ rows we get $n_o$ ciphertexts, each containing one element of the result.

Based on this description, we can derive the following performance characteristics for the naïve method:
• The total cost is \( n_o \) SIMD scalar multiplications, \( n_o \cdot \log_2 n \) rotations (automorphisms) and \( n_o \cdot \log_2 n \) SIMD additions.

• The noise grows from \( \eta \) to \( \eta \cdot \eta_{\text{mult}} \cdot n + \eta_{\text{rot}} \cdot (n - 1) \) where \( \eta_{\text{mult}} \) is the multiplicative noise growth factor for SIMD multiplication and \( \eta_{\text{rot}} \) is the additive noise growth for a rotation. This is because the one SIMD multiplication turns the noise from \( \eta \mapsto \eta \cdot \eta_{\text{mult}} \), and the sequence of rotations and additions grows the noise as follows:

\[
\eta \cdot \eta_{\text{mult}} \mapsto (\eta \cdot \eta_{\text{mult}}) \cdot 2 + \eta_{\text{rot}} \mapsto (\eta \cdot \eta_{\text{mult}}) \cdot 4 + \eta_{\text{rot}} \cdot 3 \mapsto \ldots
\]

which gives us the above result.

• Finally, this process produces \( n_o \) many ciphertexts each one containing just one component of the result.

This last fact turns out to be an unacceptable efficiency barrier. In particular, the total network bandwidth becomes quadratic in the input size and thus contradicts the entire rationale of using PAHE for linear algebra. Ideally, we want the entire result to come out in packed form \textit{in a single ciphertext} (assuming, of course, that \( n_o \leq n \)).

A final subtle point that needs to noted is that if \( n \) is not a power of two, then we can continue to use the same rotations as before, but all slots except the first slot leak information about partial sums. For example consider the case when \( n = 5 \) and \( n_i = 4 \). After two rotations the first slot contains the sum of all the elements, while the second slot only contains the sum of elements 2 to 4. We therefore \textit{must} add a random number to these slots to destroy this extraneous information about the partial sums.

\textbf{The Diagonal Method.} The diagonal method as described in the work of Halevi and Shoup [75] (and implemented in [74]) provides another potential solution to the problem of a large number of output ciphertexts. The key high-level idea is to arrange the matrix elements in such a way that after the SIMD scalar multiplications,
Figure 5-7: The naïve method is illustrated on the left and the diagonal method of Halevi and Shoup [75] is illustrated on the right. The entries in a single color live in the same ciphertext. The key feature of the diagonal method is that no two elements of the matrix that influence the same output element appear with the same color.
“interacting elements” of the matrix-vector product never appear in a single ciphertext. Here, “interacting elements” are the numbers that need to be added together to obtain the final result. The rationale is that if this happens, we never need to add two numbers that live in different slots of the same ciphertexts, thus avoiding ciphertext rotation.

To do this, we encode the diagonal of the matrix into a vector which is then SIMD scalar multiplied with the input vector. The second diagonal (namely, the elements $W_{0,1}, W_{1,2}, \ldots, W_{n_w-1,0}$) is encoded into another vector which is then SIMD scalar multiplied with a rotation (by one) of the input vector, and so on. Finally, all these vectors are added together to obtain the output vector in one shot.

The cost of the diagonal method is:

- The total cost is $n_i$ SIMD scalar multiplications, $n_i - 1$ rotations (automorphisms), and $n_i - 1$ SIMD additions.

- The noise grows from $\eta$ to $(\eta + \eta_{rot}) \cdot \eta_{mult} \times n_i$ which, for the parameters we use, is larger than that of the naïve method, but much better than the naïve method with output packing. Roughly speaking, the reason is that in the diagonal method, since rotations are performed before scalar multiplication, the noise growth has a $\eta_{rot} \cdot \eta_{mult}$ factor whereas in the naïve method, the order is reversed resulting in a $\eta_{mult} + \eta_{rot}$ factor.

- Finally, this process produces a single ciphertext that has the entire output vector in packed form already.

In our setting (and we believe in most reasonable settings), the additional noise growth is an acceptable compromise given the large gain in the output length and the corresponding gain in the bandwidth and the overall run-time. Furthermore, the fact that all rotations happen on the input ciphertexts prove to be very important for an optimization of [76] we describe in section 5.7.3, called “hoisting”, which lets us amortize the cost of many input rotations.
A Hybrid Approach. One issue with the diagonal approach is that the number of \textbf{Perm} is equal to \( n_i \). In the context of FC layers \( n_o \) is often much lower than \( n_i \) and hence it is desirable to have a method where the \textbf{Perm} is close to \( n_o \). Our hybrid scheme achieves this by combining the best aspects of the naïve and diagonal schemes. We first extended the idea of diagonals for a square matrix to squat rectangular weight matrices as shown in Figure 5-7 and then pack the weights along these extended diagonals into plaintext vectors. These plaintext vectors are then multiplied with \( n_o \) rotations of the input ciphertext similar to the diagonal method. Once this is done we are left with a single ciphertext that contains \( n/n_o \) chunks each contains a partial sum of the \( n_o \) outputs. We can proceed similar to the naïve method to accumulate these using a “rotate-and-sum” algorithm.

We implement an input packed variant of the hybrid method and the performance and noise growth characteristics (following a straightforward derivation) are described in Table 5.2. We note that hybrid method trades off hoistable input rotations in the Diagonal method for output rotations on distinct ciphertexts (which cannot be “hoisted out”). However, the decrease in the number of input rotations is multiplicative while the corresponding increase in the number of output rotations is the logarithm of the same multiplicative factor. As such, the hybrid method almost always outperforms the Naive and Diagonal methods. We present detailed benchmarks over a selection of matrix sizes in Table 5.8.

5.7 More Techniques for Homomorphic Matrix-Vector Multiplication

In this section, we describe output packing, input packing and hoisting, three other techniques that Gazelle utilizes.
5.7.1 Output Packing

The very first thought to mitigate the ciphertext blowup issue we just encountered is to take the many output ciphertexts and somehow pack the results into one. Indeed, this can be done by (a) doing a SIMD scalar multiplication which zeroes out all but the first coordinate of each of the out ciphertexts; (b) rotating each of them by the appropriate amount so that the numbers are lined up in different slots; and (c) adding all of them together.

Unfortunately, this results in unacceptable noise growth. The underlying reason is that we need to perform two serial SIMD scalar multiplications (resulting in an \( \eta_{\text{mult}}^2 \) factor; see Table 5.2). For most practical settings, this noise growth forces us to use ciphertext moduli that are larger 64 bits, thus overflowing the machine word. This necessitates the use of a Double Chinese Remainder Theorem (DCRT) representation similar to [67] which substantially slows down computation. Instead we use an algorithmic approach to control noise growth allowing the use of smaller moduli and avoiding the need for DCRT.

5.7.2 Input Packing

Before moving on to more complex techniques we describe an orthogonal approach to improve the naïve method when \( n_i \ll n \). The idea is to pack multiple copies of the input into a single ciphertext. This allows us better utilization of the slots by computing multiple outputs in parallel.

In detail we can (a) pack \( n/n_i \) many different rows into a single plaintext vector; (b) pack \( n/n_i \) copies of the input vector into a single ciphertext; and (c) perform the rest of the naïve method as-is except that the rotations are not applied to the whole ciphertext but block-by-block (thus requiring \( \log(n_i) \) many rotations). Roughly speaking, this achieves communication and computation as if the number of rows of the matrix were \( n'_o = (n_o \times n_i)/n \) instead of \( n_o \). When \( n_i \ll n \), we have \( n'_o \ll n_o \).
5.7.3 The Hoisting Optimization (Halevi and Shoup)

The hoisting optimization reduces the cost of the ciphertext rotation when the same ciphertext must be rotated by multiple shift amounts. The idea, roughly speaking, is to "look inside" the ciphertext rotation operation, and hoist out the part of the computation that would be common to these rotations and then compute it only once thus amortizing it over many rotations. It turns out that this common computation involves computing the $\text{NTT}^{-1}$ (taking the ciphertext to the coefficient domain), followed by a $w_{\text{relin}}$-bit decomposition that splits the ciphertext $[(\log_2 q)/w_{\text{relin}}]$ ciphertexts and finally takes these ciphertexts back to the evaluation domain using separate applications of NTT. The parameter $w_{\text{relin}}$ is called the relinearization window and represents a tradeoff between the speed and noise growth of the Perm operation. This computation, which we denote as $\text{PermDecomp}$, has $\Theta(n \log n)$ complexity because of the number theoretic transforms. In contrast, the independent computation in each rotation, denoted by $\text{PermAuto}$, is a simple $\Theta(n)$ multiply and accumulate operation. As such, hoisting can provide substantial savings in contrast with direct applications of the Perm operation and this is also borne out by the benchmarks in Table 5.7.

5.7.4 Additional Implementation Details

This section clarifies two important implementation details. First, recall that in order to enable faster NTT, our parameter selection requires $n$ to be a power of two. As a result the permutation group we have access to is the group of half rotations $(C_{n/2} \times C_2)$, i.e. the possible permutations are compositions of rotations by up to $n/2$ for the two $n/2$-sized segments, and swapping the two segments. The packing and diagonal selection in the hybrid approach are modified to account for this by adapting the definition of the extended diagonal to be those entries of $W$ that would be multiplied by the corresponding entries of the ciphertext when the above Perm operations are performed as shown in Figure 5-8. Finally, as described in section 5.3 we control the noise growth in SIMDSmMult using plaintext windows for the weight matrix $W$. 
5.8 Fast Homomorphic Convolutions

We now move on the implementation of homomorphic kernels for Conv layers. Analogous to the description of FC layers we will start with simpler (and correspondingly less efficient) techniques before moving on to our final optimized implementation. In our setting, the server has access to a plaintext filter and it is then provided encrypted input images, which it must homomorphically convolve with its filter to produce encrypted output images. As a running example for this section we will consider a \((f_w, f_h, c_i, c_o)\)-Conv layer with the “same” padding scheme, where the input is specified by the tuple \((w_i, h_i, c_i)\). In order to better emphasize the key ideas, we will split our presentation into two parts: first we will describe the SISO case, i.e. \((c_i = 1, c_o = 1)\) followed by the more general case where we have multiple input and output channels, a subset of which may fit within a single ciphertext.

**Padded SISO.** As seen in section 5.2, same style convolutions require that the input be zero-padded. As such, in this approach, we start with a zero-padded version of the input with \((f_w - 1)/2\) zeros on the left and right edges and \((f_h - 1)/2\) zeros on the top and bottom edges. We assume for now that this padded input image is
small enough to fit within a single ciphertext i.e. \((w_i + f_w - 1) \cdot (h_i + f_h - 1) \leq n\) and is mapped to the ciphertext slots in a raster scan fashion. We then compute \(f_w \cdot f_h\) rotations of the input and scale them by the corresponding filter coefficient as shown in Figure 5-9. Since all the rotations are performed on a common input image, they can benefit from the hoisting optimization. Note that similar to the naïve matrix-vector product algorithm, the values on the periphery of the output image leak partial products and must be obscured by adding random values.

**Packed SISO.** While the above technique computes the correct 2D-convolution it ends up wasting \((w_i + f_w - 1) \cdot (h_i + f_h - 1) - w_i \cdot h_i\) slots in zero padding. If either the input image is small or if the filter size is large, this can amount to a significant overhead. We resolve this issue by using the ability of our PAHE scheme to multiply different slots with different scalars when performing SIMDScMult. As a result, we can pack the input tightly and generate \(f_w \cdot f_h\) rotations. We then multiply these rotated ciphertexts with punctured plaintexts which have zeros in the appropriate locations as shown in Figure 5-10. Accumulating these products gives us a single ciphertext that, as a bonus feature, contains the convolution result without any leakage of partial information.

Finally, we note that the construction of the punctured plaintexts does not depend on either the encrypted image or the client key information and as such, the server can precompute these values once for multiple clients. We summarize these results in
Now that we have seen how to compute a single 2D-convolution we will look at the more general multi-channel case.

**Single Channel per Ciphertext.** The straightforward approach for handling the multi-channel case is to encrypt the various channels into distinct ciphertexts. We can then SISO convolve these $c_i$-ciphertexts with each of the $c_o$ sets of filters to generate $c_o$ output ciphertexts. Note that although we need $c_o \cdot c_i \cdot f_h \cdot f_w$ SIMDAdd and SIMDScMult calls, just $c_i \cdot f_h \cdot f_w$ many Perm operations on the input suffice, since the rotated inputs can be reused to generate each of the $c_o$ outputs. Furthermore, each these rotation can be hoisted and hence we require just $c_i$ many PermDecomp calls and $c_i \cdot f_h \cdot f_w$ many PermAuto calls.

**Channel Packing** Similar to input-packed matrix-vector products, the computation of multi-channel convolutions can be further sped up by packing multiple channels in a single ciphertext. We represent the number of channels that fit in a single ciphertext by $c_n$. Channel packing allows us to perform $c_n$-SISO convolutions in parallel in

![Figure 5-10: Packed SISO Convolution. (Zeros in the punctured plaintext shown in white.)](image)
a SIMD fashion. We maximize this parallelism by using Packed SISO convolutions which enable us to tightly pack the input channels without the need for any additional padding.

For simplicity of presentation, we assume that both $c_i$ and $c_o$ are integral multiples of $c_n$. Our high level goal is to then start with $c_i/c_n$ input ciphertexts and end up with $c_o/c_n$ output ciphertexts where each of the input and output ciphertexts contains $c_n$ distinct channels. We achieve this in two steps: (a) convolve the input ciphertexts in a SISO fashion to generate $(c_o \cdot c_i)/c_n$ intermediate ciphertexts that contain all the $c_o \cdot c_i$-SISO convolutions and (b) accumulate these intermediate ciphertexts into output ciphertexts.

Since none of the input ciphertexts repeat an input channel, none of the intermediate ciphertexts can contain SISO convolutions corresponding to the same input channel. A similar constraint on the output ciphertexts implies that none of the intermediate ciphertexts contain SISO convolutions corresponding to the same output. In particular, a potential grouping of SISO convolutions that satisfies these constraints is the diagonal grouping. More formally the $k^{th}$ intermediate ciphertext in the diagonal grouping contains the following ordered set of $c_n$-SISO convolutions:

$$\{ (|k/c_i| \cdot c_n + l, (k \mod c_i)/c_n \cdot c_n + ((k + l) \mod c_n)) \mid l \in [0, c_n) \}$$

where each tuple $(x_o, x_i)$ represents the SISO convolution corresponding to the output channel $x_o$ and input channel $x_i$. Given these intermediate ciphertexts, one can generate the output ciphertexts by simply accumulating the $c_o/c_n$-partitions of $c_i$ consecutive ciphertexts. We illustrate this grouping and accumulation when $c_i = c_o = 8$ and $c_n = 4$ in Figure 5-11. Note that this grouping is very similar to the diagonal style of computing matrix vector products, with single slots now being replaced by entire SISO convolutions.

Since the second step is just a simple accumulation of ciphertexts, the major computational complexity of the convolution arise in the computation of the intermediate
Figure 5-11: Diagonal Grouping for Intermediate Ciphertexts ($c_i = c_o = 8$ and $c_n = 4$)

ciphertexts. If we partition the set of intermediate ciphertexts into $c_n$-sized rotation sets (shown in grey in Figure 5-11), we see that each of the intermediate ciphertexts is generated by different rotations of the same input. This observation leads to two natural approaches to compute these intermediate ciphertexts.

**Input Rotations.** In the first approach, we generate $c_n$ rotations of the every input ciphertext and then perform Packed SISO convolutions on each of these rotations to compute all the intermediate rotations required by $c_o/c_n$ rotation sets. Since each of the SISO convolutions requires $f_w \cdot f_h$ rotations, we require a total of $(c_n \cdot f_w \cdot f_h - 1)$ rotations (excluding the trivial rotation by zero) for each of the $c_i/c_n$ inputs. Finally we remark that by using the hoisting optimization we compute all these rotations by performing just $c_i/c_n$ PermDecomp operations.

**Output Rotations.** The second approach is based on the realization that instead of generating $(c_n \cdot f_w \cdot f_h - 1)$ input rotations, we can reuse $(f_w \cdot f_h - 1)$ rotations...
Table 5.4: Comparing multi-channel 2D-convolutions

<table>
<thead>
<tr>
<th>PermDecomp</th>
<th>Perm</th>
<th>#in_ct</th>
<th>#out_ct</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Channel per CT</td>
<td>$c_i$</td>
<td>$(f_w f_h - 1) \cdot c_i$</td>
<td>$c_i$</td>
</tr>
<tr>
<td>Input Rotations</td>
<td>$\frac{c_i}{c_n}$</td>
<td>$(c_n f_w f_h - 1) \cdot \frac{c_i}{c_n}$</td>
<td>$\frac{c_i}{c_n}$</td>
</tr>
<tr>
<td>Output Rotations</td>
<td>$\left(1 + \frac{(c_n - 1) \cdot c_o}{c_n}\right) \frac{c_i}{c_n}$</td>
<td>$(f_w f_h - 1 + \frac{(c_n - 1) \cdot c_o}{c_n}) \frac{c_i}{c_n}$</td>
<td>$\frac{c_i}{c_n}$</td>
</tr>
</tbody>
</table>

in each rotation-set to generate $c_n$ convolutions and then simply rotate $(c_n - 1)$ of these to generate all the intermediate ciphertexts. This approach then reduces the number of input rotations by factor of $c_n$ while requiring $(c_n - 1)$ for each of the $(c_i \cdot c_o)/c_n^2$ rotation sets. Note that while $(f_w \cdot f_h - 1)$ input rotations per input ciphertext can share a common PermDecomp each of the output rotations occur on a distinct ciphertext and cannot benefit from hoisting.

We summarize these numbers in Table 5.4. The choice between the input and output rotation variants is an interesting trade-off that is governed by the size of the 2D filter. This trade-off is illustrated in more detail with concrete benchmarks in section 5.9. Finally, we remark that similar to the matrix-vector product computation, the convolution algorithms are also tweaked to work with the half-rotation permutation group and use plaintext windows to control the scalar multiplication noise growth.

### 5.8.1 More Techniques for Homomorphic Convolution

**Strided Convolutions**

We handle strided convolutions by decomposing the strided convolution into a sum of simple convolutions each of which can be handled as above. We illustrate this case for $f_w = f_h = 3$ and $s_x = s_y = 2$ in Figure 5-12.

**Low-noise Batched Convolutions**

We make one final remark on a potential application for padded SISO convolutions. Padded SISO convolutions are computed as a sum of rotated versions of the input images multiplied by corresponding constants $f_{x,y}$. The coefficient domain represen-
Decomposing a strided convolutions into simple convolutions \((f_w = f_h = 3 \text{ and } s_x = s_y = 2)\)

As a result, the noise growth factor is \(\eta_{\text{mult}} = f_{x,y} \cdot \sqrt{n}\) as opposed to \(p \cdot \sqrt{n}\), consequently noise growth depends only on the value of the filter coefficients and not on the size of the plaintext space \(p\). The direct use of this technique precludes the use of channel packing since the filter coefficients are channel dependent. One potential application that can mitigate this issue is when we want to classify a batch of multiple images. In this context, we can pack the same channel from multiple classifications allowing us to use a simple constant filter. This allows us to trade-off classification latency for higher throughput. Note however that similar to padded SISO convolutions, this has two problems: (a) it results in lower slot utilization compare to packed approaches, and (b) the padding scheme reveals the size of the filter.

### 5.9 Implementation and Micro-benchmarks

Next we describe the implementation of the Gazelle framework starting with the chosen cryptographic primitives (5.9.1). We then describe our evaluation test-bed (5.9.2) and finally conclude this section with detailed micro-benchmarks (5.9.3) for all the operations to highlight the individual contributions of the techniques described in the previous sections.
5.9.1 Cryptographic Primitives

Gazelle needs two main cryptographic primitives for neural network inference: a packed additive homomorphic encryption (PAHE) scheme and a secure 2PC scheme. Parameters for both schemes are selected for a 128-bit security level. For the PAHE scheme we instantiate the BFV scheme [29, 58], with \( n = 2048 \), 20-bit plaintext modulus, 60-bit ciphertext modulus and \( \sigma = 4 \) according to the analysis of Section 5.4.

For the 2PC framework, we use Yao’s Garbled circuits [158]. The main reason for choosing Yao over Boolean secret sharing schemes (such as the GMW protocol [70] and its derivatives) is that the constant number of rounds results in good performance over long latency links. Our garbling scheme is an extension of the one presented in JustGarble [21] which we modify to also incorporate the Half-Gates optimization [160]. We base our Oblivious Transfer (OT) implementation on the classic Ishai-Kilian-Nissim-Petrank (IKNP) [83] protocol from libOTe [129]. Since we use 2PC for implementing the ReLU, MaxPool and FHE-2PC transformation gadget, our circuit garbling phase only depends on the neural network topology and is independent of the client input. As such, we move it to the offline phase of the computation while the OT Extension and circuit evaluation is run during the online phase of the computation.

5.9.2 Evaluation Setup

All benchmarks were generated using c4.xlarge Amazon Web Services (AWS) instances which provide a 4-threaded execution environment (on an Intel Xeon E5-2666 v3 2.90GHz CPU) with 7.5GB of system memory. Our experiments were conducted using Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-1041-aws) and our library was compiled using GCC 5.4.0 using the ‘-O3’ optimization setting and enabling support for the Intel AES-NI instruction set. Our schemes are evaluated in the LAN setting similar to previous work with both instances in the us-east-1a availability zone.
Table 5.5: Fast Reduction for NTT and Inv. NTT

<table>
<thead>
<tr>
<th>Operation</th>
<th>Fast Reduction</th>
<th>Naive Reduction</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t (μs)</td>
<td>cyc/bfly</td>
<td>t (μs)</td>
</tr>
<tr>
<td>NTT (q)</td>
<td>57</td>
<td>14.68</td>
<td>393</td>
</tr>
<tr>
<td>Inv. NTT (q)</td>
<td>54</td>
<td>13.90</td>
<td>388</td>
</tr>
<tr>
<td>NTT (p)</td>
<td>43</td>
<td>11.07</td>
<td>240</td>
</tr>
<tr>
<td>Inv. NTT (p)</td>
<td>38</td>
<td>9.78</td>
<td>194</td>
</tr>
</tbody>
</table>

Table 5.6: FHE Microbenchmarks

<table>
<thead>
<tr>
<th>Operation</th>
<th>Fast Reduction</th>
<th>Naive Reduction</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t (μs)</td>
<td>cyc/slot</td>
<td>t (μs)</td>
</tr>
<tr>
<td>KeyGen</td>
<td>232</td>
<td>328.5</td>
<td>952</td>
</tr>
<tr>
<td>Encrypt</td>
<td>186</td>
<td>263.4</td>
<td>621</td>
</tr>
<tr>
<td>Decrypt</td>
<td>125</td>
<td>177.0</td>
<td>513</td>
</tr>
<tr>
<td>SIMDAdd</td>
<td>5</td>
<td>8.1</td>
<td>393</td>
</tr>
<tr>
<td>SIMDScMult</td>
<td>10</td>
<td>14.7</td>
<td>388</td>
</tr>
<tr>
<td>PermKeyGen</td>
<td>466</td>
<td>659.9</td>
<td>1814</td>
</tr>
<tr>
<td>Perm</td>
<td>268</td>
<td>379.5</td>
<td>1740</td>
</tr>
<tr>
<td>PermDecomp</td>
<td>231</td>
<td>327.1</td>
<td>1595</td>
</tr>
<tr>
<td>PermAuto</td>
<td>35</td>
<td>49.6</td>
<td>141</td>
</tr>
</tbody>
</table>

5.9.3 Micro-benchmarks

In order to isolate the impact of the various techniques and identify potential optimization opportunities, we first present micro-benchmarks for the individual operations.

Arithmetic and PAHE Benchmarks. We first benchmark the impact of the faster modular arithmetic on the NTT and the homomorphic evaluation run-times. Table 5.5 shows that the use of a pseudo-Mersenne ciphertext modulus coupled with lazy modular reduction improves the NTT and inverse NTT by roughly 7×. Similarly Barrett reduction for the plaintext modulus improves the plaintext NTT runtimes by more than 5×.

These run-time improvements are also reflected in the performance of the primitive homomorphic operations as shown in Table 5.6.
Table 5.7: Permutation Microbenchmarks

<table>
<thead>
<tr>
<th># windows</th>
<th>PermKeyGen</th>
<th>Key Size</th>
<th>PermAuto</th>
<th>Noise</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t (μs)</td>
<td>kB</td>
<td>t (μs)</td>
<td>bits</td>
</tr>
<tr>
<td>3</td>
<td>466</td>
<td>49.15</td>
<td>35</td>
<td>29.3</td>
</tr>
<tr>
<td>6</td>
<td>925</td>
<td>98.30</td>
<td>57</td>
<td>19.3</td>
</tr>
<tr>
<td>12</td>
<td>1849</td>
<td>196.61</td>
<td>100</td>
<td>14.8</td>
</tr>
</tbody>
</table>

Table 5.7 demonstrates the noise performance trade-off inherent in the permutation operation. Note that an individual permutation after the initial decomposition is roughly 8-9× faster than a permutation without any pre-computation. Finally we observe a linear growth in the run-time of the permutation operation with an increase in the number of windows, allowing us to trade off noise performance for run-time if few future operations are desired on the permuted ciphertext.

**Linear Algebra Benchmarks.** Next we present micro-benchmarks for the linear algebra kernels. In particular we focus on matrix-vector products and 2D convolutions since these are the operations most frequently used in neural network inference. Before performing these operations, the server must perform a one-time client-independent setup that pre-processes the matrix and filter coefficients. In contrast with the offline phase of 2PC, this computation is NOT repeated per classification or per client and can be performed without any knowledge of the client keys. In the following results, we represent the time spent in this amortizable setup operation as $t_{\text{setup}}$. Note that $t_{\text{offline}}$ for both these protocols is zero.

The matrix-vector product that we are interested in corresponds to the multiplication of a plaintext matrix with a packed ciphertext vector. We first start with a comparison of three matrix-vector multiplication techniques:

1. **Naive**: Every slot of the output is generated independently by computing an inner-product of a row of the matrix with ciphertext column vector.

2. **Diagonal**: Rotations of the input are multiplied by the generalized diagonals from the plaintext matrix and added to generate a packed output.
Table 5.8: Matrix Multiplication Microbenchmarks

<table>
<thead>
<tr>
<th></th>
<th>#in_rot</th>
<th>#out_rot</th>
<th>#mac</th>
<th>t_{online}</th>
<th>t_{setup}</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048x1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>7.9</td>
<td>16.1</td>
</tr>
<tr>
<td>D</td>
<td>2047</td>
<td>0</td>
<td>2048</td>
<td>383.3</td>
<td>3326.8</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>8.0</td>
<td>16.2</td>
</tr>
<tr>
<td>1024x128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>1280</td>
<td>128</td>
<td>880.0</td>
<td>1849.2</td>
</tr>
<tr>
<td>D</td>
<td>1023</td>
<td>1024</td>
<td>2048</td>
<td>192.4</td>
<td>1662.8</td>
</tr>
<tr>
<td>H</td>
<td>63</td>
<td>4</td>
<td>64</td>
<td>16.2</td>
<td>108.5</td>
</tr>
<tr>
<td>1024x16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>160</td>
<td>16</td>
<td>110.3</td>
<td>231.4</td>
</tr>
<tr>
<td>D</td>
<td>1023</td>
<td>1024</td>
<td>2048</td>
<td>192.4</td>
<td>1662.8</td>
</tr>
<tr>
<td>H</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>7.8</td>
<td>21.8</td>
</tr>
<tr>
<td>128x16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>112</td>
<td>16</td>
<td>77.4</td>
<td>162.5</td>
</tr>
<tr>
<td>D</td>
<td>127</td>
<td>128</td>
<td>2048</td>
<td>25.4</td>
<td>206.8</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>5.3</td>
<td>10.5</td>
</tr>
</tbody>
</table>

3. **Hybrid**: Use the diagonal approach to generate a single output ciphertext with copies of the output partial sums. Use the naive approach to generate the final output from this single ciphertext.

We compare these techniques for the following matrix sizes: 2048 x 1, 1024 x 128, 128 x 16. For all these methods we report the online computation time and the time required to setup the scheme in milliseconds. Note that this setup needs to be done exactly once per network and need not be repeated per inference. The naive scheme uses a 20bit plaintext window (w_{pt}) while the diagonal and hybrid schemes use 10bit plaintext windows. All schemes use a 7bit relinearization window (w_{relin}).

Finally we remark that our matrix multiplication scheme is extremely parsimonious in the online bandwidth. The two-way online message sizes for all the matrices are given by (w + 1) * c_{tsz} where c_{tsz} is the size of a single ciphertext (32 kB for our parameters).
Table 5.9: Hybrid Matrix Multiplication Window Sizing

<table>
<thead>
<tr>
<th>w_{pt}</th>
<th>w_{relin}</th>
<th>t_{online}</th>
<th>Speedup</th>
<th>t_{setup}</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048x1</td>
<td>20</td>
<td>3.6</td>
<td>2.2</td>
<td>5.7</td>
<td>2.9</td>
</tr>
<tr>
<td>1024x128</td>
<td>10</td>
<td>9</td>
<td>14.2</td>
<td>1.1</td>
<td>87.2</td>
</tr>
<tr>
<td>1024x16</td>
<td>10</td>
<td>7</td>
<td>7.8</td>
<td>1.0</td>
<td>21.5</td>
</tr>
<tr>
<td>128x16</td>
<td>20</td>
<td>20</td>
<td>2.5</td>
<td>2.1</td>
<td>3.7</td>
</tr>
</tbody>
</table>

Table 5.10: Convolution Microbenchmarks

<table>
<thead>
<tr>
<th>Input (WxH, C)</th>
<th>Filter (WxH, C)</th>
<th>Algorithm</th>
<th>t_{online} (ms)</th>
<th>t_{setup} (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(28 x 28, 1)</td>
<td>(5 x 5, 5)</td>
<td>I</td>
<td>14.4</td>
<td>11.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O</td>
<td>9.2</td>
<td>11.4</td>
</tr>
<tr>
<td>(16 x 16, 128)</td>
<td>(1 x 1, 128)</td>
<td>I</td>
<td>107</td>
<td>334</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O</td>
<td>110</td>
<td>226</td>
</tr>
<tr>
<td>(32 x 32, 32)</td>
<td>(3 x 3, 32)</td>
<td>I</td>
<td>208</td>
<td>704</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O</td>
<td>195</td>
<td>704</td>
</tr>
<tr>
<td>(16 x 16, 128)</td>
<td>(3 x 3, 128)</td>
<td>I</td>
<td>767</td>
<td>3202</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O</td>
<td>704</td>
<td>3312</td>
</tr>
</tbody>
</table>

5.9.4 Tuning the Relinearization Window

As seen in Section 5.6 the online time for the matrix multiplication operation can be improved further by a judicious selection of the window sizes based on the size of the matrix used. Table 5.9 shows the potential speed up possible from optimal window sizing. Note that although this optimal choice reduces the online run-time, the relinearization keys for all the window sizes must be sent to the server in the initial setup phase.

Next we compare the two techniques we presented for 2D convolution: input rotation (I) and output rotation (O) in Table 5.10. We present results for four convolution sizes with increasing complexity. Note that the 5 x 5 convolution is strided convolution with a stride of 2. All results are presented with a 10bit w_{pt} and a 8bit w_{relin}.

As seen from Table 5.10, the output rotation variant is usually the faster variant since it reuses the same input multiple times. Larger filter sizes allow us to save more
rotations and hence experience a higher speed-up, while for the $1 \times 1$ case the input rotation variant is faster. Finally, we note that in all cases we pack both the input and output activations using the minimal number of ciphertexts.

**Square, ReLU and MaxPool Benchmarks.** We round our discussion of the operation micro-benchmarks with the various activation functions we consider. In the networks of interest, we come across two major activation functions: Square and ReLU. Additionally we also benchmark the MaxPool layer with $(2 \times 2)$-sized windows.

For square pooling, we implement a simple interactive protocol using our additively homomorphic encryption scheme. For ReLU and MaxPool, we implement a garbled circuit based interactive protocol. The results for both are presented in Table 5.11.

### Table 5.11: Activation and Pooling Microbenchmarks

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Outputs</th>
<th>$t_{\text{offline}}$ (ms)</th>
<th>$t_{\text{online}}$ (ms)</th>
<th>$\text{BW}_{\text{offline}}$ (MB)</th>
<th>$\text{BW}_{\text{online}}$ (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>2048</td>
<td>0.5</td>
<td>1.4</td>
<td>0</td>
<td>0.093</td>
</tr>
<tr>
<td>ReLU</td>
<td>1000</td>
<td>89</td>
<td>15</td>
<td>5.43</td>
<td>1.68</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>551</td>
<td>136</td>
<td>54.3</td>
<td>16.8</td>
</tr>
<tr>
<td>MaxPool</td>
<td>1000</td>
<td>164</td>
<td>58</td>
<td>15.6</td>
<td>8.39</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>1413</td>
<td>513</td>
<td>156.0</td>
<td>83.9</td>
</tr>
</tbody>
</table>

5.10 Network Benchmarks and Comparison

Next we compose the individual layers from the previous sections and evaluate complete networks. For ease of comparison with previous approaches, we report runtimes and network bandwidth for MNIST and CIFAR-10 image classification tasks. We segment our comparison based on the CNN topology. This allows us to clearly demonstrate the speedup achieved by Gazelle as opposed to gains through network redesign.
### Table 5.12: MNIST Benchmark

<table>
<thead>
<tr>
<th>Framework</th>
<th>Runtime (s)</th>
<th>Communication (MB)</th>
<th>Offline</th>
<th>Online</th>
<th>Total</th>
<th>Offline</th>
<th>Online</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Offline</td>
<td>Online</td>
<td>Total</td>
<td>Offline</td>
<td>Online</td>
<td>Total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecureML</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A</td>
<td>4.7</td>
<td>0.18</td>
<td>4.88</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MiniONN</td>
<td>0.9</td>
<td>0.14</td>
<td>1.04</td>
<td>3.8</td>
<td>12</td>
<td>47.6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gazelle</td>
<td>0</td>
<td>0.03</td>
<td>0.03</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CryptoNets</td>
<td>-</td>
<td>-</td>
<td>297.5</td>
<td>-</td>
<td>-</td>
<td>372.2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>0.88</td>
<td>0.4</td>
<td>1.28</td>
<td>3.6</td>
<td>44</td>
<td>15.8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MiniONN</td>
<td>0</td>
<td>0.03</td>
<td>0.03</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gazelle</td>
<td>0</td>
<td>0.03</td>
<td>0.03</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DeepSecure</td>
<td>-</td>
<td>-</td>
<td>9.67</td>
<td>-</td>
<td>-</td>
<td>791</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>1.34</td>
<td>1.36</td>
<td>2.7</td>
<td>7.8</td>
<td>5.1</td>
<td>12.9</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Chameleon</td>
<td>0.15</td>
<td>0.05</td>
<td>0.20</td>
<td>5.9</td>
<td>2.1</td>
<td>8.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gazelle</td>
<td>0.481</td>
<td>0.33</td>
<td>0.81</td>
<td>47.5</td>
<td>22.5</td>
<td>70.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>3.58</td>
<td>5.74</td>
<td>9.32</td>
<td>20.9</td>
<td>636.6</td>
<td>657.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ExPC</td>
<td>-</td>
<td>-</td>
<td>5.1</td>
<td>-</td>
<td>-</td>
<td>501</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gazelle</td>
<td>0.481</td>
<td>0.33</td>
<td>0.81</td>
<td>47.5</td>
<td>22.5</td>
<td>70.0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

#### The MNIST Dataset.
MNIST is a basic image classification task where we are provided with a set of $28 \times 28$ grayscale images of handwritten digits in the range $[0 - 9]$. Given an input image our goal is to predict the correct handwritten digit it represents. We evaluate this task using four published network topologies which use a combination of FC and Conv layers:

**A** 3-FC with square activation from [110].

**B** 1-Conv and 2-FC with square activation from [68].

**C** 1-Conv and 2-FC with ReLU activation from [133].

**D** 2-Conv and 2-FC with ReLU and MaxPool from [99].

Runtime and the communication required for classifying a single image for these four networks are presented in table 5.12.

For all four networks we use a 10bit $w_{pt}$ and a 9bit $w_{relin}$.

Networks A and B use only the square activation function allowing us to use a much simpler AHE base interactive protocol, thus avoiding any use of GCs. As such we only need to transmit short ciphertexts in the online phase. Similarly our use
The CIFAR-10 Benchmark

<table>
<thead>
<tr>
<th>Framework</th>
<th>Runtime (s)</th>
<th>Communication (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Offline</td>
<td>Online</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MiniONN</td>
<td>472</td>
<td>72</td>
</tr>
<tr>
<td>Gazelle</td>
<td>9.34</td>
<td>3.56</td>
</tr>
</tbody>
</table>

of the AHE based FC and Conv layers as opposed to multiplications triples results in 5-6× lower latency compared to [99] and [110] for network A. The comparison with [68] is even more the stark. The use of AHE with interaction acting as an implicit bootstrapping stage allows for aggressive parameter selection for the lattice based scheme. This results in over 3 orders of magnitude savings in both the latency and the network bandwidth.

Networks C and D use ReLU and MaxPool functions which we implement using GC. However even for these the network our efficient FC and Conv implementation allows us roughly 30× and 17× lower runtime when compared with [128] and [99] respectively. Furthermore we note that unlike [128] our solution does not rely on a trusted third party.

The CIFAR-10 Dataset. The CIFAR-10 task is a second commonly used image classification benchmark that is substantially more complicated than the MNIST classification task. The task consists of classifying 32 × 32 color with 3 color channels into 10 classes such as automobiles, birds, cats, etc. For this task we replicate the network topology from [99] to offer a fair comparison. We use a 10bit \( w_{pt} \) and a 8bit \( w_{relin} \).

We note that the complexity of this network when measure by the number of multiplications is 500× that used in the MNIST network from [133], [128]. By avoiding the need for multiplication triples Gazelle offers a 50× faster offline phase and a 20× lower latency per inference showing that our results from the smaller MNIST networks scale to larger networks.
5.11 Conclusions and Future Work

In conclusion, this work presents Gazelle, a low-latency framework for secure neural network inference. Gazelle uses a judicious combination of PAHE and garbled circuit based 2PC to obtain $20 - 30 \times$ lower latency and $2.5 - 88 \times$ lower online bandwidth while providing similar or better security guarantees when compared with multiple recent 2PC-based state-of-art secure network inference solutions [99, 110, 128, 133], and more than 3 orders of magnitude lower latency and 2 orders of magnitude lower bandwidth than purely homomorphic approaches [68]. We briefly recap the key contributions of our work that enable this improved performance:

1. Selection of prime moduli that simultaneously allow SIMD operations, low noise growth and division-free and lazy modular reduction.

2. Avoidance of ciphertext-ciphertext multiplications to reduce noise growth.

3. Use of secret-sharing and interaction to emulate a lightweight bootstrapping procedure allowing us to evaluate deep networks composed of many layers.

4. Homomorphic linear algebra kernels that make efficient use of the automorphism structure enabled by a power-of-two slot-size.

5. Sparing use of garbled circuits limited to ReLU and MaxPool functions with linear-size Boolean circuits.

6. A compact garbled circuit-based transformation gadget that allows us to securely compose the PAHE-based and garbled circuit based layers.
Chapter 6

Homomorphic Protocols for faster Matrix Multiplication and Vector-OLE

6.1 Matrix Multiplication

Many data-analytics and machine learning algorithms use matrix multiplication as an integral building block. Let us consider two concrete examples to motivate the need for efficient secure matrix multiplication algorithms.

The first examples comes from the domain of secure neural network inference that we saw in Chapter 5. The the evaluation of fully-connected layers in a neural network can directly be represented by a matrix-vector product i.e. if the input activations are represented by the vector \( \mathbf{x} \in \mathbb{R}^n \), layer weights and biases are represent by the matrix \( \mathbf{W} \in \mathbb{R}^{m \times n} \) and vector \( \mathbf{b} \in \mathbb{R}^m \), then the output activations are given by \( \mathbf{y} = \mathbf{Wx} + \mathbf{b} \). When we consider the batched version of the same problem, i.e., when we try to infer multiple input images using the same model we observe that the fully-connected layer equation is transformed to its natural matrix analog \( \mathbf{Y} = \mathbf{WX} + \mathbf{B} \). In fact, this analysis can also be extended to the other linear layers such as convolutions using a standard sparse Toeplitz matrix formulation.
The second example comes from the domain of training linear regression models. Given a data-set \( \mathbf{X} \in \mathbb{R}^{n \times d} \) containing \( n \)-entries of \( d \)-covariates and the corresponding output vector \( \mathbf{y} \in \mathbb{R}^n \), linear regression aims to find the best fit parameters \( \beta \) and \( \gamma \) s.t.

\[
\mathbf{y} = \mathbf{X}\beta + \gamma
\]

The classic solution to this problem computes \( \hat{\beta} = (\hat{\mathbf{X}}^T\hat{\mathbf{X}})^{-1}\hat{\mathbf{X}}^T\mathbf{y} \), where \( \hat{\mathbf{X}} \) corresponds to the matrix formed by appending a column of ones to \( \mathbf{X} \) and \( \hat{\beta} \) is the best estimate for \( (\beta, \gamma) \). Most often this computation is implemented using an iterative gradient descent algorithm 1. Line number 5 requires the computation of an matrix multiplication. In the case where either the covariates or the data-set entries are shared across multiple parties this matrix multiplication must be performed securely.

**Algorithm 1** Linear Regression

1: procedure LINREG(\( \hat{\mathbf{X}}, \mathbf{y}, \alpha, k \))
2: \( i \leftarrow 1 \)
3: \( \hat{\beta} \leftarrow 0 \)
4: for \( i \leq k \) do
5: \( \hat{\beta} \leftarrow \hat{\beta} - \frac{\alpha}{n} (\hat{\mathbf{X}}^T\hat{\mathbf{X}}\hat{\beta} - \hat{\mathbf{X}}^T\mathbf{y}) \)
6: \( i \leftarrow i + 1 \)
7: end for
8: end procedure

As such we present an efficient solution for the task of secure matrix multiplication which we define below,

**Definition 6.1.1.** (Secure Matrix Multiplication) Secure matrix multiplication is a two party functionality where the parties picks matrices \( \mathbf{A}_1 \in \mathbb{Z}_q^{m \times k} \) and \( \mathbf{A}_2 \in \mathbb{Z}_q^{k \times n} \) and at the end of the functionality receive matrices \( \mathbf{B}_i \in \mathbb{Z}_q^{m \times n} \) such that \( \mathbf{B}_1 + \mathbf{B}_2 = \mathbf{A}_1\mathbf{A}_2 \).

Matrix multiplication is a classic algorithm that has been studied and optimized over many years. In practice two algorithms are most commonly used. The first is the \( O(n^3) \) school book approach with independently computes each entry in the final matrix as an inner-product. While the asymptotic complexity is known to be
$O(n^{2.373})$ [156], as the matrices grow larger the $O(n^{\log_2 7})$ Strassen algorithm is often preferred due to the large constants in the former approach. Even when using Strassen’s algorithm, the $O(n^3)$ algorithm is used as the base-case for the recursion. Thus it is instructive to look at the performance of this simple work-horse algorithm when instantiated using a PAHE scheme.

In the following sections we simplify the presentation by assuming that $m = k = n$ where $n$ is also the number of slots supported by the PAHE scheme we use.

### 6.1.1 Baseline Approach

The simplest approach to computing the matrix product $A_1 A_2$ would involved encrypting the columns of $A_2$ and sending them over to the first party who could then evaluate the $n^2$ inner-products. Each inner-product evaluation would then require a single SIMDScMult and $\log n$ Perm operations. Since each Perm operation is itself an $O(n \log n)$ operation, the total complexity of the algorithm is $O(n^3 \log^2 n)$ which severely degrades performance.

### 6.1.2 Using Gazelle

This issue of reducing the number of Perm operations was also observed in matrix-vector multiplication implemented in Gazelle. We recall that our optimal solution in that case amortized the $O(n \log n)$ Perm operation to generate the $n$-outputs by using the diagonal approach. Moreover the hoisting optimization allowed the quasi-linear part of the PermDecomp operation to be shared across $n$-outputs. Thus this approach for computing the complete matrix multiplication requires an $O(n^2 \log n)$ hoisting phase followed by an additional $O(n^3)$ PermAuto overhead. Concretely PermAuto has a constant overhead proportional to the number of relinearization windows used. For typical parameters this varies between $3 - 8$. 

129
6.1.3 Outer-Product Matrix Multiplication

Instead we propose a new outer-product based matrix multiplication strategy that avoids the need for any Perm operations altogether. In this case we exploit the fact that multiplying a column vector by a row vector generates a partial product matrix without the need for any Perm operations. We can then simply accumulate these partial product matrices in a SIMD fashion to generate the final output matrix. For sufficiently large matrices this alleviates the need for any Perm operations and ensures a $O(n^3)$ computational complexity for the secure variant. Moreover we are also able to avoid the constant PermAuto overhead from Gazelle.

6.1.4 Implementation

We implemented this matrix multiplication on top of the Gazelle code-base. Table 6.1 compares the new approach with the repeated application of the matrix-vector multiplication from Gazelle and the plaintext-version. As seen below our approach is upto 20× computationally faster than Gazelle and just 3× slower than plaintext for large matrices.

<table>
<thead>
<tr>
<th>(m,k,n)</th>
<th>Gazelle</th>
<th>Proposed</th>
<th>Plaintext</th>
</tr>
</thead>
<tbody>
<tr>
<td>(128, 128, 128)</td>
<td>KeyGen</td>
<td>11</td>
<td>7.6</td>
</tr>
<tr>
<td></td>
<td>Client</td>
<td>64</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td>Server</td>
<td>494</td>
<td>150</td>
</tr>
<tr>
<td>(2048, 128, 2048)</td>
<td>KeyGen</td>
<td>121</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>Client</td>
<td>1024</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td>Server</td>
<td>30740</td>
<td>1550</td>
</tr>
</tbody>
</table>

Table 6.2 presents the end-to-end runtime of our approach including network overhead when compared with the state-of-art. We observe that our approach is both computationally faster and consumes lower bandwidth when compared with [100]. As a result we outperform them both over fast and slow networks.
Table 6.2: Comparison of Online Matrix Multiplication

<table>
<thead>
<tr>
<th>(m,k,n)</th>
<th>Runtime</th>
<th>Communication</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Proposed Speedup</td>
<td>Proposed Speedup</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>(128, 128, 128)</td>
<td>2.21 s 0.3 s 7.4</td>
<td>8.25 MB 0.5 MB 16</td>
<td>128, 128, 128</td>
<td>100 Mbps</td>
<td>10 Mbps</td>
<td>100 Mbps</td>
<td>10 Mbps</td>
</tr>
<tr>
<td>(1024, 128, 1024)</td>
<td>71.4 s 6.7 s 11</td>
<td>8.25 MB 0.5 MB 16</td>
<td>1024, 128, 1024</td>
<td>100 Mbps</td>
<td>10 Mbps</td>
<td>100 Mbps</td>
<td>10 Mbps</td>
</tr>
</tbody>
</table>

6.2 Vector-OLE

In Chapter 5 we have seen Yao's Garbled Circuit protocol as one potential option for implementing interactive 2PC. The main advantage of Yao's protocol is that the number of rounds in the protocol is always 2 and thus independent of the function being evaluated. This is often advantageous when running the an interactive 2PC long latency links. On the flip-side the online bandwidth used is proportional to the security parameter. For a typical 128-bit security application this can often be the bottleneck in the online performance. In these cases, the GMW protocol [70] offers a compelling alternative. Like Yao, the GMW protocol also represents the function that we want to evaluate securely as a boolean circuit. It then evaluates this circuit layer by layer. Thus the total number of rounds is now proportional to the depth of the circuit. However the main advantage is that the online bandwidth is no longer dependent on the security parameter. Thus, in low latency scenarios such as when the two parties are connected over a LAN, GMW often outperforms Yao [137]. The layer by layer evaluation inherent in the GMW protocol is based on the repeated use of a primitive called boolean multiplication triples.

**Definition 6.2.1.** (Boolean Multiplication Triple) A multiplication triple is a two party functionality where each party picks a bit $a_i$ and at the end of the functionality receives a bit $b_i$ such that $a_1 + a_2 = b_1 \cdot b_2$.

The most common way to instantiate the multiplication triple functionality is using the OT functionality.

**Definition 6.2.2.** (Oblivious Transfer) OT is a two party functionality where the
first party inputs a bit $x$ and the second party inputs two bits $(y_0, y_1)$. At the end of the functionality the first party receives the bit $y_x = y_0 + x \cdot (y_0 + y_1)$.

It is then easy to see that the tuples $(x, y_x)$ and $(y_0, y_0 + y_1)$ indeed represent a boolean multiplication triple, thus showing us a way to construct the multiplication triple functionality from OT.

A natural extension of OT is Oblivious Linear Evaluation (OLE) where the elements are picked from a general field $F$ instead of $F_2$. Analogous to the use of OT in instantiating the GMW protocol for binary circuits, one can use OLE to instantiate a variant for arithmetic circuits i.e. a circuit whose wires are field elements and the gates represent addition and multiplication in $F$ [84].

Vector-OLE simply refers to the SIMD variant of the OLE functionality.

**Definition 6.2.3. (Vector OLE)** Vector-OLE is a two party functionality where the first party picks an element $x$ from $F$ and the second party picks two vectors $a$ and $b$ from the space $F^n$. At the end of the functionality the first party receives the vector $ax + b$.

In addition to "arithmetic-GMW", Vector-OLE has also been used in arithmetic garbling [9, 11] and nearest-neighbor style algorithms [54] for face-recognition.

In some applications such as the offline phase of MiniONN [99], a parallel variant of Vector-OLE called Batch-OLE is useful.

**Definition 6.2.4. (Batch OLE)** Batch-OLE is a two party functionality where the first party picks a vector $x$ from $F$ and the second party picks two vectors $a$ and $b$ from the space $F^n$. At the end of the functionality the first party receives the vector $a \cdot x + b$ where `·' represent the element-wise product.

Both Vector-OLE and Batch-OLE can be represented as a depth-1 circuits that can be easily evaluated using an PAHE scheme. The key fact however is that due to simplicity of the circuit at hand any overhead required to enforce the function-privacy of the PAHE scheme has a significant impact on concrete performance. As such we next present a detailed analysis of the standard approach to implementing
function-private Vector-OLE. Based on this analysis we identify key bottlenecks and implement an optimized version of these foundational 2PC primitives.

### 6.2.1 Baseline Approach

Figure 6.2.1 shows a baseline protocol for implementing Vector-OLE using a PAHE scheme. The standard approach to guarantee function privacy involves the addition of a randomized-encryption of zero followed by noise flooding. Noise-flooding in itself involves sampling a new noise term from a wide Gaussian ($\chi_{\text{flood}}$).

<table>
<thead>
<tr>
<th>Baseline Vector-OLE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client</strong></td>
</tr>
<tr>
<td>$s, \epsilon \leftarrow \chi, a \leftarrow R_q$</td>
</tr>
<tr>
<td>$\text{pk} \leftarrow (a, a \cdot s + \epsilon)$</td>
</tr>
<tr>
<td><strong>Server</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Online</strong></td>
</tr>
<tr>
<td>$x \leftarrow R_p$</td>
</tr>
<tr>
<td>$e_1 \leftarrow \chi, a_1 \leftarrow R_q$</td>
</tr>
<tr>
<td>$\text{ct} \leftarrow (a_1, a_1 \cdot s_1 + x\Delta + e_1)$</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>$y \leftarrow \frac{[\text{ct}<em>{1,y} - \text{ct}</em>{0,y}]}{\Delta}$</td>
</tr>
</tbody>
</table>

We now analyze how the parameters must be set for a secure instantiation of the above Vector-OLE scheme to better understand these implementation issues. In order to work with a plaintext modulus $p$ and initial noise bound $B$, we need to set the final..
ciphertext modulus $q$ such that it can allow for correct decryption after evaluating Vector-OLE. The noise growth in the Vector-OLE is dominated by the multiplication step, where the post multiplication noise can be bounded by $B' = (B + r)p\xi$ where $n$ is the number of slots allowed by the PAHE scheme and $r = q \mod p$. In general a random selection of $q$ and $p$ can result in $r$ being as large as $p$. On the other hand typically the initial noise is much smaller ($B \ll p$). Hence we have $B' < 2Bp^2n$. Additionally in order to maintain a statistical security of $2^{-\lambda}$ bits we need to flood this final ciphertext with noise having standard deviation $2^{n}nB' = 2^{\lambda+1}Bp^2n^2$. Since $q$ must be large enough to support both the plaintext and the noise we require $q > 2^{\lambda+1}Bp^2n^2$. Thus, even a modest plaintext modulus of 40-bits with a similar statistical security parameter requires a ciphertext modulus that is close to 190-bits. Since this is larger than the native 64-bit word size on modern machines and we need to use a three-limb DCRT representation of the ciphertexts to minimize the number of big-integer operations. In the above example a we need to sample from the flooding Gaussian distribution with a roughly 140-bit standard deviation. This noise must hence be first sampled as a big-integer and then transformed into the DCRT representation which is slow.

6.2.2 Optimizing Bandwidth

The first observation we make is that when instantiating Batch-OLE the total bandwidth the above scheme requires is $2n \log q$-bits in either direction. When compared with an insecure plaintext implementation that which requires $n \log p$-bits either direction we have a net overhead of $2 \log q/\log p$. A similar asymptotic analysis for Vector-OLE results in the same net overhead. Plugging in the formulae from the previous section shows a concrete overhead of $6 + 2(\lambda + 1 + \log(Bn^2))/(\log p)$ which in the case of our example is already as high as $9.5 \times$. We note that this is already very competitive with prior work, e.g. [10] implements a Vector-OLE scheme with a concrete overhead of $10 \times$ for similar parameters. We now use the following two techniques to improve the communication bandwidth of the PAHE based Vector-OLE. Since $a_1$ in the first message to the server is a random element in $R_q$ we can simply replace it
with seed to a PRNG. Second we notice that most of the bits on the return path from the server to the client are purely noise. As such we can use modulus reduction [28] to reduce the size of $ct_y$. Apriori one might hope to reduce the ciphertext modulus almost down to the size the plaintext modulus. However modulus reduction itself adds some noise due to the rounding operation. This additional noise is bounded by $n \|s\|_\infty$. Thus we can hope to achieve asymptotic overheads of $2 + \frac{2 \log(n \|s\|_\infty)}{\log p}$ for Vector-OLE and $1 + \frac{\log q}{2 \log p} + \frac{\log(n \|s\|_\infty)}{2 \log p}$ for Batch-OLE. Using a ternary distribution for the noise distribution we can set $\|s\|_\infty = 1$. For the practical setting of our running example this results in a communication overhead of $2.6 \times$ and $3.7 \times$ for Vector-OLE and Batch-OLE respectively.

### 6.2.3 Parameter selection for bigint-free rounding

We next remark on the details of the underlying implementation of the modulus reduction functionality. Consider the DCRT representation of a ciphertext using the ciphertext modulus-set $(q_1, \ldots, q_n)$. At the end of homomorphic computation the noise in the ciphertext grows to fill the available space inside a ciphertext. If the plaintext fits within a single ciphertext limb say $\mod q_1$, then [28] shows us how to reduce the DCRT ciphertext to a single limb Single Chinese Remainder Theorem (SCRT) ciphertext. The procedure essentially requires us to round the coefficient representation of the ciphertext modulo the product of the rest of the primes. The key issue is that when using multiple ciphertext moduli this the coefficient representation is too large to fit within the native word-size and this reduction requires a big-integer operation. The resulting big-integer is much slower than native division and in the case of our running example we observe that this simple big-integer rounding operation dominates roughly half of the online runtime.

We propose to address this issue through a clever selection of the ciphertext primes. In addition to the correctness and security constraints from Chapter 5 we additionally impose the following three constraints on the ciphertext moduli:

1. $|q_i \mod p|$ is small: This ensures that the additional noise generated by the
rounding operation is small.

2. $|\prod q_i \mod p|$ is small: This ensures that the wraparound noise generated during multiplication is small.

3. $q_i = 2^{i-1}(q - 1) + 1$: This ensures a simple approximate rounding algorithm that can performed directly using the CRT representation modulo the primes $q_i$.

We summarize our rounding algorithm in the case of two and three limbs in the following lemmas. In both cases the rounding is either exact or off by one. In our use-case we can simply absorb this rounding error into the final noise.

**Lemma 6.2.1.** *(Two Limb Reduction)* Given the CRT representation of a number $x = (x_1, x_2)$ modulo the primes $(q, 2q - 1)$,

$$\frac{x}{q_1} \approx (2(x_2 - x_1)) \mod q_2$$

**(6.1)**

**Lemma 6.2.2.** *(Three Limb Reduction)* Given the CRT representation of a number $x = (x_1, x_2, x_3)$ modulo the primes $(q, 2q - 1, 4q - 3)$,

$$\frac{x}{q_1q_2} \approx ((4 \cdot 3q_3^{-1} - 2) \cdot x_1 - 4x_2 + (8 \cdot 3q_3^{-1}) \cdot x_3) \mod q_3$$

**(6.2)**

Finally we select each of these three primes such that they support the efficient modular multiplication algorithm presented in [6]. As a result of this parameter selection we can preserve the efficient NTT from Chapter 5 while simultaneously speeding up the rounding algorithm by more than $10\times$. Overall this speeds up the online runtime of the Vector-OLE by a factor of two.

### 6.2.4 Rounding based function privacy

The baseline Vector-OLE scheme that we described floods the resultant ciphertext $ct_m$ with noise to maintain function privacy. We then perform modulus reduction to reduce the size of the ciphertext. Since the noise that we added in the previous step
lives in the least significant bits of the ciphertext it seems redundant and wasteful to first add a large noise and then to round it off in the immediate next step. Hence we next analyze the necessity of flooding the ciphertext with Gaussian noise. A Learning with Errors (LWE) ciphertext for the BFV scheme [58] looks as follows:

\[
\text{ct} = (a, a \cdot s + m\Delta + e) = (a_{Q_1} \cdot Q_1 + \hat{a}_{Q_1}, (a_{Q_1} \cdot Q_1 + \hat{a}_{Q_1}) \cdot s + m\Delta + e)
\]

(6.3)

(6.4)

where, \(Q_i = \prod_{n \neq i} q_n\), \(a_{Q_i} = \lfloor a/Q_i \rfloor\), \(\hat{a}_{Q_i} = a \mod Q_i\) and \(\Delta = \lfloor \Pi a/p \rfloor\). Rounding this ciphertext by \(Q_1\) gives us:

\[
\text{ct}_{\text{rnd}} = \left( a_{Q_1}, a_{Q_1} \cdot s + \left\lfloor \frac{\hat{a}_{Q_1} \cdot s + m\Delta + e}{Q_1} \right\rfloor \right)
\]

(6.5)

Now, the decryption equation for \(\text{ct}_{\text{rnd}}\) is given by \(\text{ct}_{\text{ct}} - \text{ct}_{\text{ct[0]}}\) where \(\Delta_{q_1} = \left\lfloor \frac{q_1}{p} \right\rfloor\). Thus we must ensure that the additional noise introduced by the rounding does not violate the correctness of decryption. The noise in \(\text{ct}_{\text{rnd}}\) can be given by,

\[
\eta_{\text{ct}_{\text{rnd}}} = \left\lfloor \frac{\hat{a}_{Q_1} \cdot s + m\Delta + e}{Q_1} \right\rfloor - m\Delta_{q_1}
\]

(6.6)

\[
= \left\lfloor \frac{\hat{a}_{Q_1} \cdot s + m \left( \frac{a_{Q_1}}{p} - \frac{Q_e}{p} \right) + e}{Q_1} \right\rfloor - m\Delta_{q_1}
\]

(6.7)

\[
= \left\lfloor \frac{m q_1}{p} + \frac{\hat{a}_{Q_1} \cdot s - m \frac{Q_e}{p} + e}{Q_1} \right\rfloor - m\Delta_{q_1}
\]

(6.8)

\[
= \left\lfloor \frac{mq_1}{p} - \frac{mQ_e}{pQ_1} + \frac{\hat{a}_{Q_1} \cdot s + e}{Q_1} \right\rfloor
\]

(6.9)

Thus we see that as long as the original error \(e \ll Q_1\) it can only affect the least significant bit of the new error with probability \(e/Q_1\). Thus we replace noise flooding
with just the rounding operation and pick parameters to ensure $e2^\lambda < Q_1$. This gives us $\lambda$-bit statistical security.

### 6.2.5 NTT optimization

As seen in Chapter 5, the implementation of a PAHE scheme primarily involves $O(n)$ algorithms with the exception of the NTT operation which requires $O(n \log n)$ time. Thus the final optimization we perform relates to reducing the number of NTT calls required to implement Vector-OLE. Given a DCRT ciphertext modulus with $m$-limbs, the number of NTT calls required for the various operation are presented in Table 6.3.

The baseline scheme requires the receiver to compute 1 PAHE.Enc($sk, x$) and 1 PAHE.Dec, while the sender needs to compute 2 Encode operations, 1 PAHE.Enc($pk, x$), 1 SIMDAdd and 1 SIMDScMult and 1 Round. Thus we require $m + 3$ NTT for the receiver and $7m + 5$ NTT for the sender. Our first observation is that instead of looking at all these operations individually one can combine them and perform a global optimization on the number of NTT. For example, although we need to transform $\beta$ from the plaintext evaluation representation to the plaintext coefficient representation we do not need to perform the $m$ NTT to transform it to the final evaluation representation. This is because we can simply add $\beta$ during the rounding step when we transform the product ciphertext back into the coefficient representation. A similar analysis also holds true for the Gaussian error terms sampled during the public key encryption of zero. As such we can reduce the final NTT count to $4m + 4$ for the sender. For our running example this results in 6 and 16 NTT for the receiver and

<table>
<thead>
<tr>
<th>Operation</th>
<th>Number of NTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAHE.Enc($sk, x$)</td>
<td>$m + 1$</td>
</tr>
<tr>
<td>PAHE.Enc($pk, x$)</td>
<td>$3m + 1$</td>
</tr>
<tr>
<td>Encode($x$)</td>
<td>$m + 1$</td>
</tr>
<tr>
<td>SIMDAdd($ct_1, ct_2$)</td>
<td>0</td>
</tr>
<tr>
<td>SIMDScMult($ct, x$)</td>
<td>0</td>
</tr>
<tr>
<td>Round($ct$)</td>
<td>$2m + 2$</td>
</tr>
<tr>
<td>PAHE.Dec($sk, ct$)</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6.3: Number of NTT calls required per operation
sender respectively. A final observation is that by sending the output ciphertext $c_{xy}$ in the coefficient representation we can reduce the sender NTT count by 2 at the cost of a single addition transform at the receiver.

### 6.2.6 Implementation

We implement the above scheme on top of our 2PC framework from Chapter 5. We use a 42-bit plaintext modulus and a 60, 61 and 62-bit ciphertext modulus chain. To maintain 128-bit computational security for the effective 183-bit ciphertext modulus we need to set the slot-size to 8192. We observe a single 8192-point NTT takes 200μs. The complete scheme has a latency of 5ms when evaluating 8192-wide Vector and Batch-OLE. When compared with [10] this results in 8× lower computation and 4× lower communication.
Chapter 7

Conclusion and Future Work

7.1 Summary and Contributions

This thesis attempts to address new security challenges that have arisen with the rapid growth in low-cost IoT sensing solutions and increased adoption in cloud computing. We have focused on two specific problems. The first is the issue of authentication in presence of physical attacks such as side-channel attacks and fault injection attacks. The second is the issue of allowing two parties to compute on shared functions of their private data. The main contribution of this thesis is the realization of multiple integrated secure systems. We now summarize these contributions and our learnings and present some avenues for future work.

Chapter 2, presented a wireless authentication tag using FeCaps NVDFFs for security applications and demonstrated mitigation techniques against passive and active threat models. The tag performs per-query key updates before each protocol invocation to prevent side-channel attacks. The NVDFF key storage and FeCaps-based energy backup solution enabled complete key save-restore functionality for safe shutdown against power-glitch attacks. Our proposed authentication tag provides a secure proof of origin in a highly globalized supply chain.

Our approach for protecting against these attacks takes a very holistic view. We do not treat the underlying process technology as just a means for implementing cryptographic accelerators. Similarly we do not treat our accelerators as just a means
of implementing high-level protocols. Instead we co-design our accelerators and protocols to leverage unique analog-domain features provided by the underlying process. A key-outcome of this process is the understanding that such a unified approach can place less demanding requirements on the cryptographic accelerators themselves. For example the Keccak accelerator described in Chapter 2 must only be secure against SPA as opposed to the more sophisticated DPA attacks.

Chapter 3 presented a low-resource ECC accelerator, which was combined with a detuning based power delivery front-end to build a secure wireless charging solution. Authenticating the charger using a public key-based scheme allowed a scalable solution that avoided the need for the receiver to securely store a private key. The use detuning allowed us to forego any switched passive components, which typically need to be implemented with switches capable of tolerating large amounts of current and/or voltage stress. This allowed us to achieve up to 16× power blocking from unauthenticated chargers and completely invert the power distribution profile in a one charger-two receiver scenario despite a 4:1 distance asymmetry between the receivers.

The key takeaway from this project is the rejection of conventional wisdom that public-key cryptography is too expensive for energy-harvesting applications. We demonstrate how to design an ECC accelerator that is the same order of magnitude in size as state-of-art AES implementations [103]. The impact of this observation on real-world security cannot be overstated. Practical side-channel attacks have been demonstrated against firmware-verification implementations [132] where the secret primitives like AES were used. The use of public-key primitives would make this attacks impossible since the devices would no longer store the secret signing key.

Chapter 4, improved upon these previous two solutions by combining their best features. We developed a new leakage resilient zero-knowledge ID scheme based on the SXDH assumption in bilinear pairing groups on the BN Elliptic Curves. We demonstrated the practicality of this protocol in an embedded environment by implementing it on a low-end RISCV [154] processor by reusing traditional cryptographic accelerators aimed for TLS. This allowed us to quantify the concrete implementation

142
cost of deploying a provably secure scheme and we demonstrated real-time prover run-time with sub-1mW power consumption.

This work is one of the first concrete steps in translating provable side-channel resistant protocols to resource constrained embedded systems. More importantly, we demonstrate that practical performance can be achieved by reusing hardware used for more conventional protocols. This allows system designers to implement and evaluate these new protocols without having to pay an upfront cost for dedicated silicon and we envision that such a design strategy will have a positive impact on the adoption of these new schemes.

Chapter 5 presented Gazelle, a scalable and low-latency system for secure neural network inference, using an intricate combination of homomorphic encryption and traditional two-party computation techniques (such as garbled circuits). We evaluated our protocols on benchmark neural networks trained on the MNIST and CIFAR-10 datasets and obtained 20–30× lower latency and 2.5–88× lower online bandwidth when compared with multiple recent 2PC-based state-of-art secure network inference solutions [99, 110, 128, 133], and more than 3 orders of magnitude lower latency and 2 orders of magnitude lower bandwidth than purely homomorphic approaches [68].

While the Gazelle is a software implementation of a secure computation protocol, it inherently represents a hardware designer's outlook to tackling this problem. A careful analysis of the various costs, both communication and computation, guides our partitioning and mapping the various layers to PAHE and GC implementations. We take the view that PAHE scheme is just a massively parallel SIMD machine that allows SIMD additions, multiplications and rotations. We then compile higher level functionality such as matrix-vector products and convolutions to this three instruction SIMD machine. Optimizing both the mapping of the operations to these instructions and the implementations of these instructions themselves is at the core of our performance gains. We believe that is a great opportunity to replicate this design strategy to a variety of secure computation protocols.

Finally, Chapter 6 built on Gazelle, and demonstrated how to implement secure interactive protocols for two building block algorithms. We first demonstrated a new
outer-product based protocol for matrix-matrix multiplication that is (8-25×) faster than the state-of-the art [100] and can be within a 3× overhead of plaintext computation. Second we demonstrated a new big-integer free vector-OLE protocol using a new rounding based function privacy mechanism. We showed how to select primes compatible with our new rounding based scheme and reduce concrete communication-overhead to within 2.5× of an insecure plaintext implementation. This improves the state-of-art [10] by 8× on computation and 4× on communication.

Both these examples are a continuation of the approach we employ in Gazelle. We take a deep-dive into the constituent operations required in each of these protocols. We identify the source of the bottle-necks and then optimize them to achieve the above performance gains. We finally note that although much of the work in this thesis has been described from the point of view improving runtimes, the exact same approach also improves the energy consumption of these protocols. The underlying reason is the observation that we are not simply running more threads or blindly parallelizing computation to gain performance. Instead we reducing the amount of computation required to perform these tasks. This is best exemplified by the NTT optimization described in the context of Vector-OLE in Chapter 6.

### 7.2 Future Work

Integrating and deploying this research into real-world systems will undoubtedly present new and interesting directions for future research. We now briefly summarize some of these directions for continuing the work presented in this thesis.

1. **Secure Wireless Tagging:** Chapter 2 presented a compact-integrated tagging solution using FeCap-based non-volatile memory. A natural extension would be to investigate the suitability and security of using other low-power non-volatile memory technologies like Spin Torque Tunelling (STT)-RAM [153] and Conductive Bridge (CB)-RAM [93]. The main motivation would be to further decrease the size of the on-chip passives by selecting memories that offer lower backup energy per bit. A second direction relates to the continued miniaturization of
these tags. The current tag implementation requires an external coil for near-field data and power transfer. This increases both the bulk and packaging cost of the wireless tags. One approach to solving this problem is to move to a far-field millimeter-wave setting with on-die antennae similar to [146]. The limited power conversion efficiency of these systems will motivate interesting protocol and architecture research for building secure tagging solutions.

2. **Low Resource Public-Key Cryptography**: ECC has been the work-horse of low resource public-key cryptography. A direct extension of the work in Chapter 3 is move to a higher 128-bit security level using the larger K-283 binary curve. Similarly one can also envision using the same design philosophy in the context of prime curves such as EC25519. In energy harvesting scenarios, where the application is bottlenecked by the maximum instantaneous power that can be delivered and not the total energy per authentication, an interesting alternative would be to investigate the suitability of LWE-based lattice schemes. Such an approach would have the added benefit of resilience against quantum adversaries.

3. **Provable Leakage Resilient ID**: Chapter 4 presents a real-time leakage-resilient ID scheme. We envisage the following research directions for further improving security and performance. From a theoretical standpoint it would be highly-desirable to achieve continual leakage resilience from simpler assumption such as DDH as opposed to SXDH. Failing this, one can hope to extend the work from [97] to a more practical prime-order setting as opposed to the composite-order pairing groups used therein. One observation from Chapter 4 is that, the software implementation of the pairing functionality results in a large amount of time and energy being spent in simply moving the data back and forth between the hardware accelerator and the main processor. This overhead is particularly challenging for short operations such as base field-additions which take just a single cycle on the hardware accelerator. Thus one approach to improve performance is to modify the accelerators to amortize the data movement cost.
across multiple field operations on the hardware accelerator.

4. **Secure Neural Network Inference**: Chapter 5 presents a new framework for secure neural network inference. There are a number of natural avenues to build on top of this work. The most immediate is handling larger application-specific neural networks that work with substantially larger inputs to tackle data analytics problems in the medical and financial domains. We can easily extend our techniques to a large variety of classic two-party tasks such as privacy-preserving face recognition [136] which can be factored into linear and non-linear phases of computation similar to what is done in this work. In the low-latency LAN setting, it would also be interesting to evaluate the impact of switching out the garbled-circuit based approach for a GMW-based approach which would allow us to trade off latency to substantially reduce the online bandwidth. A final, very interesting and ambitious line of work would be to build a compiler that allows us to easily express arbitrary computations and automatically factor the computation into PAHE and two-party primitives.

5. **Fast secure multi-party computation (MPC) primitives**: Both the Gazelle neural network inference protocol and the protocols from Chapter 6 are secure in a passive/semi-honest setting where both parties are expected to obey the protocol while trying to learn more about the other party's inputs. For more wide-spread adoption we should investigate protocols that are secure in the active setting.

6. **Hardware for Secure Computation**: A natural avenue for extending this work is coupling our work on faster protocols with dedicated hardware to implement these protocols. We note that most of the building blocks that would be required for such an accelerator such as symmetric encryption, hash functions and public-key encryption were already presented in this thesis. A major direction of new work would be to implement the PAHE-scheme in hardware. A key point to note is that an edge device would only need to implement the PAHE.Enc and PAHE.Dec functionality. This can greatly simplify the implementation of
these edge-devices and provide resource constrained devices a meaningful way to protect their privacy when interacting with the cloud.
Bibliography


[38] Myungjoon Choi, Taekwang Jang, Junwon Jeong, Seokhyeon Jeong, David Blaauw, and Dennis Sylvester. A current-mode wireless power receiver with optimal resonant cycle tracking for implantable systems. pages 372-373, 2016.


[48] Qingkuan Dong, Jianing Zhang, and Lili Wei. A sha-3 based rfid mutual authentica-
tion protocol and its implementation. In Signal Processing, Communication
IEEE, 2013.

[49] Dan Doughty and E Peter Roth. A general discussion of Li ion battery safety.

[50] Léo Ducas and Damien Stehlé. Sanitization of the ciphertexts. In Annual In-
ternational Conference on the Theory and Applications of Cryptographic Tech-


[53] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. Scapi:

[54] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald
Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In Interna-
tional Symposium on Privacy Enhancing Technologies Symposium, pages 235–

A security-enhanced uhf rfid tag chip. In Digital System Design (DSD), 2013

[56] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter,
Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin

[57] Junfeng Fan and Ingrid Verbauwhede. An updated survey on secure ecc im-
plementations: Attacks, countermeasures and cost. Cryptography and security,

[58] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic

[59] Vincenzo Fiore, Egidio Ragonese, Sahel Abdinia, Stephanie Jacob, Isabelle
Chartier, Romain Coppard, Arthur van Roermund, Eugenio Cantatore, and
Giuseppe Palmisano. 30.4 a 13.56 mhz rfid tag with active envelope detection
in an organic complementary tft technology. In Solid-State Circuits Conference
IEEE, 2014.


[95] Hyung-Min Lee, Chiraag S Juvekar, Joyce Kwong, and Anantha P Chandrakasan. A nonvolatile flip-flop-enabled cryptographic wireless authentication


