Title
Privacy-Preserving Computation for Nuclear Safeguards

Author
Negus, Mitchell Gardiner

Publication Date
2021

Peer reviewed|Thesis/dissertation

Privacy-Preserving Computation for Nuclear Safeguards

by

Mitchell Gardiner Negus

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Nuclear Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Rachel N. Slaybaugh, Chair
Professor Lee A. Bernstein
Associate Professor Raluca Ada Popa
Dr. David R. Farley

Fall 2021

Privacy-Preserving Computation for Nuclear Safeguards

Abstract

Privacy-Preserving Computation for Nuclear Safeguards

by

Mitchell Gardiner Negus

Doctor of Philosophy in Engineering – Nuclear Engineering

University of California, Berkeley

Professor Rachel N. Slaybaugh, Chair

Nuclear safeguards are a key component in international efforts to prevent the proliferation of nuclear weapons. The International Atomic Energy Agency (IAEA) and its nuclear inspectors are tasked with administering these safeguards measures, ensuring to the best of their ability that weaponizable nuclear material is not created or diverted without the knowledge of the international community. However, no safeguards system is invincible, and so it is necessary for the IAEA to be constantly advancing and expanding its capabilities. At the same time, despite the vast majority of the international community agreeing that the proliferation of nuclear weapons should be avoided via their commitments permitting the IAEA to administer safeguards within their jurisdictions, it is expected that some States will be hesitant to allow the agency to significantly expand the scope of implemented safeguards. This leaves large quantities of potentially insightful data unavailable to safeguards administrators evaluating compliance with international regulations.

This dissertation proposes a solution to reconcile these competing interests in the form of privacy-preserving computation. Using privacy-preserving algorithms, IAEA inspectors and IAEA Member States may realize a new paradigm where nuclear facility data can be analyzed for safeguards purposes without that data ever being revealed to anyone other than the facility operators. These algorithms open the possibility that entirely new data streams may become accessible to IAEA inspectors, especially those that lend themselves to modern data analytics techniques.

This work represents the culmination of recent efforts to showcase for the first time how nuclear safeguards could be implemented in a privacy-preserving context. It includes a proof-of-concept demonstration of how the privacy-preserving technique of garbled circuits could be used for a safeguards analysis of real nuclear data, highlighting the present feasibility of such algorithms. It also introduces *CypherCircuit*, a new software framework for building and evaluating garbled circuits that is intended to facilitate the adoption of privacy-preserving technologies by skeptical or non-expert audiences. The *CypherCircuit* Python package is designed to cultivate an understanding of garbled circuits through an intuitive, transparent, and accessible design, encouraging the application of privacy-preserving techniques to novel challenges—in this case nuclear safeguards.

To Amanda

For seeing me through.

Contents

List of Figures

List of Tables

List of Algorithms

Preface

The twenty-first century has seen data—especially "big data"—thrust into the mainstream as a highly-prized commodity. Computational memory, storage, and processing power have converged to the point where this data can be harnessed for tremendous gain. Computer algorithms, most notably those falling under the umbrella of machine learning,[1] analyze scores of data points at a scale far surpassing human capability in order to gain insights, draw conclusions, and predict the future. The results can be awe-inducing (and occasionally flat-out wrong), however the tools continue to improve and demand for them has skyrocketed.

For the most part, this is where a great deal of research focus lies: improving the tools. More accurate tools cut risks. Less biased tools reduce inequalities. Better understood tools yield insights into otherwise imperceptible patterns. Beyond these algorithmic enhancements, however, the vast majority of these tools are best improved simply by being fed more data. In fact, access to data is arguably even more important than having the best tools to analyze that data, as technological improvements continue shrinking the gap between tools on the cutting edge and those that are widely available. From that perspective, it will be those with the data to drive their analytical tools who will be able to make the best inferences and predictions. Across almost every facet of life, from social media to self-driving cars, banking to healthcare, non-profits to national governments, simply having access to volumes of data will be a key factor in maintaining technological relevance.

The assumption that data will be shared is not a given, however. Data are often generated by an entity—be it an individual, enterprise, or nation—other than the one performing the analysis, and those entities only share data when the perceived benefits gained outweigh the potential costs of sacrificing privacy. In many circumstances those consequences may seem incidental, but there are also many cases where security and privacy considerations put a damper on data sharing.

At the individual level, social media and mobile app users generate interactions that are widely tracked by the hosts for marketing.[2] Many users give away access to this information at the prospect of minor convenience, but recently passed legislation suggests that users may be directing renewed attention to locking down their data for personal security.[3] In industry, sensors at manufacturing facilities collect data on highly-specialized fabrication processes, and for a price, cloud service providers may offer to analyze this data and identify potential efficiency enhancements. Forward-thinking businesses would likely be hesitant to share information that could potentially expose their trade secrets, but with adept legal teams, contracts may be established to protect the generator's assets (data *and* trade secrets), thereby offering financial security.

[1] Machine learning is not strictly necessary, as traditional statistical analyses may also draw valuable conclusions.

[2] It is becoming common knowledge that most mobile apps are collecting troves of data about their users, but two concrete examples are given by Van Kleek et al. in *Better the Devil You Know: Exposing the Data Sharing Practices of Smartphone Apps* (2017) and Razaghpanah et al. in *Apps, Trackers, Privacy, and Regulators* (2018).

[3] Specifically the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA).

A similar dynamic exists for sovereign nations, institutions that have often recognized value in keeping information private, whether that be information related to national intelligence, military, or scientific purposes. Although most of these nations support international cooperation and abide by international agreements, they are also motivated to act in their own best interests, providing security to their citizens and businesses. They may strongly favor international regulations like those preventing the spread of nuclear weapons, but they may also be simultaneously reluctant to share more with the international community beyond what is required under international law. This reluctance is presumably (and justifiably) stronger for sovereign states than it is for businesses: exposed data may disadvantage national industries, with more limited options for seeking restitution in an international legal environment than in a stricter national system; or, even more critically, data exposure may have national security repercussions.

In each of the examples offered here, it is easy to see how sharing data may seem at odds with protecting the privacy of that data. It is thus unsurprising that harnessing the power of big data and preserving data privacy frequently compete in the public consciousness as a false dichotomy. Will those entities that value privacy fail to maximally exploit their data keep pace in a modern world? Or, will those same entities be hobbled by exploitation if they embrace the potential of modern data analytics but leave their information exposed? In fact, it may not be either.

The marvel of privacy-preserving computation is that it offers the ability for both futures—one where data can be shared freely for maximum benefit, but the privacy of that data can be guaranteed. Flavors of this idea have already emerged as the global community has wrestled with the COVID-19 pandemic, with mobile phone applications being developed to perform secure contact tracing to varying degrees of success.[4] The same idea could presumably be extended to almost any mobile application seeking to leverage user data (for location, shopping, web browsing, etc.). The opportunities seem uncountable, yet some of the most impactful arenas for these privacy-preserving data analytics are far beyond the scope of individual consumer privacy. These are technology spaces where privacy-preserving computation could be used to extend business collaboration, or even promote international cooperation. One such space, discussed thoroughly in the remainder of this dissertation, is the application of privacy-preserving data analytics to encourage the international community to accept even stronger safeguards against the proliferation of nuclear weapons. Perhaps these technologies even eventually play a role in drawing down existing nuclear weapon stockpiles.

This dissertation introduces privacy-preserving data analytics to the field of nuclear safeguards, but the greater takeaway is how privacy-preserving computation can be applied to a broad assortment of technical challenges. It is my belief that privacy-preserving computation offers a solution that can bridge the gap between a modern society driven by data and a community that respects privacy. Bearing that in mind, as you read this dissertation I encourage you to consider where privacy-preserving computation might be creatively applied in your own daily encounters.

– MITCH NEGUS

[4] See, for example, Reichert et al in *Privacy-Preserving Contact Tracing of COVID-19 Patients* (2020).

Acknowledgments

A good friend and I once shared a laugh about how, when all was said and done, the research behind a computational PhD often feels like it could have been done merely in a month or two. The reality, we realized, was that mastering tools and building skills over the process of years was the real value of the degree, far more than the results of the research. It was only later, as I finished composing this work geographically removed from the school I have called home for the past five years, that I recognized we had forgotten something. It seems obvious now, just as the code we produce to solve research problems seems obviously simple in hindsight, but the value of earning a PhD also stems from the relationships that are forged and reforged along the way. This dissertation would never have materialized without a slew of supporting characters, and so I would be misplaced if I failed to acknowledge their contributions along the way.

First and foremost, I have to thank Professor Rachel Slaybaugh. As my research advisor, she has been my advocate since before I even began my journey at Berkeley. It was with her guidance that I was able to grow my passion for doing high-quality, reproducible computational research—and pass along that passion to others. It was her patience while I grew as an aspiring researcher that kept me committed to finishing my degree. And, it was her willingness to allow me to take on this project, blending nuclear safeguards and cryptography in a context almost entirely foreign to the majority of the nuclear engineering discipline, that has launched me on what is bound to be a challenging and rewarding career.

Second, I have to express my gratitude to Dr. David Farley. From the very earliest few months in my graduate program where he began posing interesting questions about the application of novel machine-learning techniques to nuclear engineering challenges, to later proposing the fundamentals of this project and serving as my technical mentor, he has shaped my entire PhD experience and inspired me to think creatively. His willingness to help me navigate the complex world of our multi-lab, multi-thrust project has built the foundation for my post-graduate future.

Third, I have to acknowledge the contributions of Professors Lee Bernstein and Raluca Ada Popa. Both Professors took time out of their (dare-I-say busy) schedules to review my dissertation, but both played instrumental roles at other key moments in my graduate career. For Professor Bernstein, his nuclear physics class was the first class I took at UC Berkeley, and one of the few that helped me feel comfortable jumping from undergraduate studies in Physics (and no knowledge of nuclear engineering) to Berkeley's nuclear engineering department. Turning to Professor Ada Popa, she and her students offered early assistance in helping me understand the nuances of privacy-preserving computation, a field where I had almost no prior knowledge. With those contributions in mind, I think it is fitting that they have both had a hand in this final research product.

Following the research thread, I must also give credit to the MUSE team at Oak Ridge National Laboratory for letting me use their data in this project. That data allowed this work to go beyond just speculation, and actually show where the research might actually be applied in something re-

sembling a real safeguards situation.

Now, as I alluded in the introduction, there are many people that I would like to acknowledge for making this work possible, even if they have not necessarily contributed directly to the research product. Starting with those who led me down this path of pursuing a research career, I have to specifically thank Professor Scott Auerbach and Dr. Samuele Sangiorgio. It was Professor Auerbach's teaching that drove me to embrace interdisciplinary research, while also always making sure to prioritize communication, remember my audience, and convey the importance of my research before all else. It also helped that he was a Berkeley alumnus who reassured me that studying on the opposite coast from home would be worth my while. Dr. Sangiorgio was the first mentor that I met on that other coast, and who first inspired in me a love of computational research. He gave me a patient introduction to high-performance computing, in a way that let me grow at my own pace while learning more in a summer than perhaps in any semester before or since. I was both awed and enthralled, and I've been finding joy in programming ever since.

Next, I wish to recognize all those whose paths meaningfully crossed mine over the course of my graduate journey. To start, I wish to recognize the hard work that is put in daily by the front office staff of the Department of Nuclear Engineering. I am thinking specifically of Kirsten Wimple Hall and Sara Harmon, who both helped steer me through the Berkeley system and come out in one piece, but also all the staff who work behind the scenes to keep the department rolling.

Then, there are my colleagues and group mates, those who shared in the daily grind of performing computational nuclear engineering research. I have to thank Josh, for sharing my affinity for Git version control; Mario, for his advice and friendship; and Kelly, Marissa, Vanessa, and April, for always being a motivating team willing to share in the camaraderie of our adventure. There are my close friends from elsewhere in the nuclear engineering department: Franziska, for having a constant and endless sense of humor; Kathy, for always reminding me to think critically and for teaching me how to actually ski; along with Milos, Maria, Sami, Mauricio, Justin, Chris, and Rebecca for all the treasured memories. And, there are also those friends from outside the department, and in some cases outside UC Berkeley entirely. These are the members of the EECS Softball League and the Strawberry Canyon Track Club, friends who gave me a much needed escape from the seemingly all-consuming indefiniteness of graduate school.

Beyond those individuals, I am especially grateful for the handful of folks I have grown to consider my extended family. The year 2020 will go down in the history books, and a guy could not ask for better roommates with whom to weather a "three-week" shelter-in-place order than Andrew, Eric, Jack, and Sam. From board games to late night chats, horror movies to sumo, these gentlemen all made a pandemic bearable—and that's saying a lot given five people all trying to use *that* kitchen. Then, there are Tanya and Martina, who were flat out, hands down, no-questions-asked the best landlords imaginable. They welcomed me into their world so that I could study in what can only be described as paradise: a cottage fulfilling every childhood fantasy of treehouse living, tucked into a garden oasis, and lit at night by the word HOPE emblazoned high in the neighbor's redwood, a necessary reminder on many of the longest nights. I still cannot believe my luck that our paths crossed, and to this day consider it something of a miracle.

Next, I turn to Amanda, who has been by my side nearly every step of the way. I knew what we had was special when we would chat endlessly, late into the night, always being on the same page.

We are no longer drowning the department messaging platform in our discussions, but four years on, some things haven't changed too much. From *Seinfeld* to sushi, to road trips and remembering what it's like to be home, she has been my constant and biggest supporter. And, speaking of Lewises, I cannot forget to thank Linda and Mark—and not just for all the *fantastic* food—but for the company, the inspiration, and the love they have shown me over the past couple years.

Finally, I thank my family. First, to Bumpa, Grandma, and Grammy, for staying interested and invested in my research, following along each step of the way and cheering from back home. Then, to Chris, for being patient with me as I pushed my finish date further and further back; the date's not getting any later now. Although my next stop in Albuquerque is still far from home, I ought to have a bit more time on my hands. And last, but certainly not least, to Mom and Dad. To Mom, who has encouraged me to be endlessly optimistic but to never forget my roots. And to Dad, who taught me to never give up and always see the big picture, to spread my wings and fly. I could never have done this without you. I love you.

Acronyms

2PC two-party computation

ABACC Brazilian-Argentine Agency for Accounting and Control of Nuclear Materials
AmLi americium-lithium

BMR Beaver-Micali-Rogaway
BSP benign source population

CDF cumulative distribution function
CSA comprehensive safeguards agreement

DA destructive analysis
DAG directed acyclic graph
DLT distributed ledger technology
DPRK Democratic People's Republic of Korea

EDAS enhanced data authentication system
EOSS electronic optical sealing system
EWMA exponentially weighted moving average

FBOS fiber optic general purpose seal
FHE fully homomorphic encryption
FOM figure of merit

GMW Goldreich-Micali-Wigderson

HEU highly enriched uranium
HFIR High Flux Isotope Reactor
HPGe high-purity germanium detector

IAEA International Atomic Energy Agency

JSON JavaScript Object Notation
JUA joint use agreement
JUE joint use equipment

KMP key measurement point

MBA material balance area
MBR material balance report

MC&A nuclear material control and accounting
MINOS Multi-Informatics for Nuclear Operations Scenarios
MPC multiparty computation
MUSE Modeling Urban Scenarios and Experiments

NDA non-destructive analysis
NGSS next generation of surveillance system
NIST National Institute of Standards and Technology
NNDC National Nuclear Data Center
NNWS non-nuclear weapons State
NPT Treaty on the Non-Proliferation of Nuclear Weapons
NRC United States Nuclear Regulatory Commission
NWS nuclear weapons State

OLEM on-line enrichment monitor
ORNL Oak Ridge National Laboratory
OT oblivious transfer

PIMS plutonium inventory monitoring system

REDC Radiochemical Engineering Development Center
RO random oracle
ROI region of interest
RSA Rivest-Shamir-Adleman

SCR spectral comparison ratio
SFDL secure function definition language
SHDL secure hardware definition language
SNM special nuclear material
SNRI short notice random inspection
SQ significant quantity
SRA State or Regional Authority with responsibility for safeguards
SSAC State system of accounting for and control of nuclear material

TCP Transmisssion Control Protocol

U.N. United Nations
UFFM unattended fuel flow monitor
UMS unattended monitoring system

VIFF Virtual Ideal Functionality Framework
VOA voluntary offer agreement

XOR exclusive or

ZKP zero-knowledge proof

Chapter 1

Introduction

Modern developments in the field of cryptography enable advanced calculations while simultaneously protecting privacy. These state-of-the-art classes of privacy-preserving algorithms allow data to be collected and analyzed in ways that control the amount and character of information that is shared between parties. In fact, when implemented properly, these techniques may be used to produce actionable insights based on data that is kept secure, never being revealed to the analyst.

There are an unfathomable number of circumstances where these types of privacy-preserving data analytics could be used. These range from applications in consumer electronics to enterprise management and beyond. Looking internationally, in an international community seeing increasing tension between sovereign states, the concept of privacy-preserving data analytics presents an opportunity to maintain, and maybe even strengthen, international cooperation and trust. Nowhere is this international cooperation more important to the preservation of global stability than in upholding commitments to global nuclear policy. These are policies aimed at preventing the proliferation of nuclear weapons and nuclear weapons material, prohibiting the testing of nuclear weapons, and working towards nuclear weapons disarmament.

International nuclear safeguards serve as a prime example of where privacy-preserving computation may have surprising implications. Nuclear safeguards, as defined by the International Atomic Energy Agency (IAEA), are measures designed "to deter the spread of nuclear weapons by the early detection of the misuse of nuclear material or technology". Defined specifically in agreements concluded between member States and the IAEA, safeguards require States to prove that their nuclear material is being used for declared, lawful activities. These practices can include thoroughly accounting for relevant nuclear material, facilitating regular inspections, and submitting to remote monitoring.

Modern safeguards implementations often share common tools and techniques. Radiation detectors are used by both field inspectors and permanently installed remote monitoring systems. Tamper-evident seals are used to guarantee consistent nuclear material balances over time, but also to verify that equipment has not been unscrupulously modified between inspections. In all cases, safeguards are designed to provide the strongest possible assurances that nuclear material is not being diverted. In pursuit of this goal, considerable attention has been paid to creating more advanced safeguards implementations. Examples include higher resolution detectors, more comprehensive

video surveillance systems, or more secure seals. As "big data" analytical tools and methodologies have seen dramatic improvements over the past decade—and captured public attention—the demand for sophisticated safeguards driven by data analytics has also surged. New initiatives have focused on how to use the vast amount of information potentially available at nuclear facilities to bolster safeguards efforts. And, while many of these efforts have looked at how to use historically valuable and relatively common data modalities, many others have looked at "non-traditional" safeguards data (soil and water chemistry, infrasound, and micro-seismic monitoring are a few examples). These new, innovative techniques in data analytics present an opportunity for administering safeguards with unprecedented levels of confidence.

Unfortunately for nuclear regulators, however, nations and businesses have often adopted stances averse to sharing their sensitive data. In the case of nuclear safeguards, nations (and their nuclear facilities) are often naturally opposed to sharing any more than the minimum amount of information required by relevant safeguards agreements and international laws. These positions protect competitive advantages and national security, but are also likely to inhibit efforts to use this data to its fullest extent in advanced applications of safeguards. Importantly, this aversion to sharing data should not necessarily be misconstrued as a condemnation of nuclear safeguards, and presumably all involved parties agree with the its goals. A world safe from nuclear proliferation is an asset, both for nations concerned with global stability and businesses that rely on favorable nuclear policies to continue operating. This is especially true as developed nations looking to minimize carbon emissions consider the merits of adding more nuclear power to their energy portfolios. Unfortunately, it just is not perceived to be worth the risk for a State or facility to share information, even if the information is only potentially sensitive.

Privacy-preserving data analytics is an obvious solution to this dilemma. By facilitating an international regulator to reach conclusions about a State or facility's compliance with safeguards while simultaneously guaranteeing the security of that information, all parties can be satisfied. Institutions retain exclusive ownership and guaranteed privacy of their data, and the inspectorate can employ state-of-the-art data analysis to draw robust conclusions of the institution's activity. Taking this one step further, having the ability to enact safeguards in this way is an additional tool at the disposal of international nuclear negotiators, should they try to enact or extend safeguards in reluctant nations.

Despite this potential, privacy-preserving data analytics is computationally intensive and has not yet been demonstrated on large datasets of time series. Neither has it been applied to the nuclear field, though similar privacy-preserving concepts have been suggested for facilitating nuclear arms-control verification and disarmament. This dissertation demonstrates the former, and makes an attempt to motivate further efforts towards realizing the latter. First, it proposes the use of garbled circuits, a privacy-preserving protocol achieving secure multiparty computation (MPC), to analyze nuclear safeguards relevant time series data. Second, it demonstrates applications of MPC acting on real time series of nuclear radiation spectra to identify notable material transfer events. Third, it introduces a software tool—the *CypherCircuit* Python package—to facilitate comprehension and encourage adoption of garbled circuit MPC techniques, especially oriented towards potentially skeptical audiences. In more detail this dissertation is broken down into the following chapters:

- **Chapter 2: Nuclear Safeguards** – An introduction to nuclear safeguards and the international policy space surrounding their implementation frames the motivation for international cooperation on nuclear nonproliferation and articulates challenges facing the nuclear safeguards community. This chapter covers why and how the IAEA administers nuclear safeguards, framing the context within which the privacy-preserving algorithms and tools must operate.
- **Chapter 3: Privacy-Preserving Computation** – Privacy-preserving computation is reviewed, with a specific focus on garbled circuits, the protocol used in subsequent demonstrations. These garbled circuits are detailed in depth, building the foundations for understanding the garbled circuit constructions presented in subsequent chapters. A quick overview of existing state-of-the-art MPC codes is also offered to summarize existing options for performing privacy-preserving computation.
- **Chapter 4: Safeguards Anomaly Detection** – The MUSE dataset is introduced, along with a survey of algorithms that excel at identifying anomalies in time series data. From among these algorithms, two are chosen for demonstration trials where privacy-preserving garbled circuits were developed to pick out nuclear material transfers from nearly two months of radiation spectra. The trial reviews describe how the public Obliv-C engine was used for performing MPC. The results of these trials are shown to confirm that the anomaly detection algorithms successfully identify anomalous events in a datasets representative of nuclear safeguards, but without revealing information about the input data.
- **Chapter 5: *CypherCircuit*** – The garbled circuit package *CypherCircuit*—developed specifically for this work—is presented as a software package designed to facilitate the adoption of garbled circuits through understanding. Design criteria of the software, the framework's basic structure, and available optimizations are all discussed. It is shown how the *CypherCircuit* framework might act as a vehicle for training users by demonstrating garbled circuit protocols effectively, especially when the audience consists of non-expert users. Timing statistics are provided for comparing *CypherCircuit* to other state-of-the-art cryptographic research codes.
- **Chapter 6: Conclusion and Discussion** – Finally, the project is reviewed in its entirety. Thought is given to where this work could see additional growth, particularly emphasizing nuclear safeguards and nuclear non-proliferation applications.

Taken all together, it is shown that privacy-preserving computation can be used effectively for securely performing data analytics on nuclear safeguards data. It is demonstrated that garbled circuits, programmed with an appropriate anomaly detection algorithm, are able to present a regulator with successfully identified nuclear material transfer events from locally collected radiation spectra. Critically, the regulator never receives this spectral information directly. This result, when presented alongside a tool that encourages fundamental understanding of privacy-preserving computation, provides strong motivation for considering MPC as a viable option for future safeguards systems. For the nuclear safeguards community where anything but minimal risk is intolerable, anything less will fail to convince the involved parties that privacy-preserving data analytics should be adopted.

Chapter 2

Nuclear Safeguards

In his December 1953 address to the General Assembly of the United Nations, the United States President Dwight Eisenhower proposed an "international atomic energy agency" charged with facilitating the peaceful use of atomic energy [1]. In his remarks, Eisenhower recognized that nuclear research and development could yield benefits for civil society and not just serve military purposes.

In the context of the time, the concept of peaceful nuclear technology was still relatively novel. Dominating the world's perspective of nuclear technology was the rapidly escalating Cold War arms race between the United States and the USSR. The United States had dropped two atomic bombs on Hiroshima and Nagasaki in 1945, after which the Soviet Union immediately began efforts to build its own nuclear explosives.[1] By 1949 the USSR had scaled up its own research (and espionage [3]) program, demonstrating its nuclear capabilities in the RSD-1 nuclear test [4]. Great Britain followed suit, conducting a successful nuclear test of its own in 1952 [5]. Concurrently with these notable milestones, the United States (and forthwith the Soviet Union) began to amass substantial nuclear weapons stockpiles. Eisenhower's address acknowledged as much, confirming that the United States stockpile exceeded "by many times the total equivalent of the total of all bombs and all shells that came from every plane and every gun in every theatre of war in all the years of the Second World War" [1]. Estimates place American nuclear weapon stockpiles at more than 1,100 warheads in 1953, with the USSR having scaled up its fledgling arsenal to more than 100 warheads by that time [6].

Electricity from nuclear energy was also still largely relegated to the realm of science fiction. Nuclear power reactors had not yet been used to produce commercial electricity, and it was not until 1954 when the Obninsk Nuclear Power Plant came online in the USSR that a reactor was used to provide civilians with electricity [7]. Although the United States had first used the EBR-I reactor to produce electricity in 1951 [8], it was not until the passage of the Atomic Energy Act of 1954 that the government permitted significant declassification of nuclear reactor technology for commercial use [9]. In fact, civilian nuclear power in the United States was not realized until the opening of the Shippingport Atomic Power Station in May 1958.[2]

[1] Premier Joseph Stalin established the Special Committee on the Atomic Bomb on August 20, 1945, exactly two weeks after the United States dropped the first nuclear bomb on Hiroshima [2].

[2] The plant began operation in December 1957, but was not officially dedicated until May 1958 [10]. The dedication

The International Atomic Energy Agency (IAEA) was ultimately established in 1957 and, as Eisenhower had proposed, was chartered to "accelerate and enlarge the contribution of atomic energy to peace, health and prosperity throughout the world" [12]. The founding parties recognized that a mission to facilitate the peaceful use of nuclear material and technology necessitated that the agency make efforts to prevent the misuse of its services. The agency's secondary objective thus became ensuring "that assistance provided by it or at its request or under its supervision or control is not used in such a way as to further any military purpose" [12]. To do this, the agency was authorized to administer safeguards in two types of scenarios: first, in any case where its services were being used, and second, at the request of a State. Such a request might arise either from a single State looking to safeguard its own nuclear material and operations, or it might be the result of parties engaging in a related treaty or agreement that requires parties to request IAEA safeguards.

Here it is important to distinguish nuclear safeguards from other types of security. The IAEA states the the objective of nuclear safeguards is "the timely detection of diversion of significant quantities of nuclear material from peaceful nuclear activities to the manufacture of nuclear weapons or of other nuclear explosive devices or for purposes unknown, and deterrence of such diversion by the risk of early detection" [13]. Specifically, safeguards are intended to prevent nuclear material from being channeled towards weaponization, by either state or non-state actors. On the contrary, nuclear security is often framed in terms of protection of the health and safety of the public against explicit and immediate potential threats. The United States Nuclear Regulatory Commission (NRC) identifies threats like theft and sabotage as those that would be combatted by nuclear security programs [14]. A successful terrorist attack on a nuclear power plant could present grave consequences to the general public, but would not *necessarily* result in any significant diversions of weaponizable nuclear material. Security programs should therefore be designed to protect facilities against terrorist sabotage. Conversely, a nation that transfers nuclear material from its legitimate and peaceful nuclear operations to a covert, illicit weapons program may not present an (immediate) threat to the health and safety of the public; it would, however, be violating international protocol. Safeguards exist to dissuade and detect a rogue State's endeavors.

This chapter offers a broad background on nuclear safeguards, establishing the foundations of the international nuclear regulatory system. The proposed application of privacy-preserving technology presented in the following chapters is intended to enhance this safeguards regime, and so must reflect the goals of the safeguards system while operating within its scope.

2.1 The International Safeguards Arena

As an international organization bound by its statute and the will of the global community, the IAEA does not possess the authority to unilaterally impose safeguards on its member States. Instead, the IAEA is tasked with administering safeguards in the two scenarios mentioned previously: whenever it is providing assistance towards advancing peaceful uses of nuclear technology, and when it is requested to provide safeguards by its members [12]. While these restrictions superficially appear to limit the IAEA's purview, international nonproliferation treaties have leveraged the latter

featured Eisenhower waving a ceremonial "neutron wand" over a neutron counter to supposedly activate the plant [11].

option to broadly expand the ability of the IAEA to safeguard nuclear material. Such treaties exist both globally and regionally, and they *require* that States party to the treaty conclude safeguards agreements with the IAEA [15: Article III]. In effect, to remain compliant with the obligations set forth in those nonproliferation treaties, States party to the treaty must request and accept IAEA safeguards.

2.1.1 Treaties and International Policy

Among all international nuclear treaties, the Treaty on the Non-Proliferation of Nuclear Weapons (NPT) is the most significant driver of nuclear safeguards. The treaty, which entered into force on March 5, 1970, was the culmination of nearly a decade of international cooperation [15]. Nonproliferation efforts had been steadily gaining momentum, with widespread affirmation of the principle showcased in a 1961 resolution passed unanimously by the United Nations (U.N.) General Assembly [16, 17].[3]

Though it consists of eleven articles, the treaty is often characterized as a three-pillar system: nonproliferation, promotion of peaceful nuclear uses, and disarmament [17]. States party to the treaty are identified as either nuclear weapons States (NWSs) or non-nuclear weapons States (NNWSs), with a NWS defined in Article IX as any States which had "manufactured and exploded a nuclear weapon or other nuclear explosive device prior to 1 January, 1967" [15]. As of 2021, the treaty had 191 States party to the treaty, with only the United States, the Russian Federation, the United Kingdom, France, and China being recognized as NWSs. Notable non-parties include India, Pakistan, Israel, and South Sudan, all countries who are either believed to possess nuclear weapons capabilities, or in the case of South Sudan, not widely regarded as a proliferation risk. Additionally, the Democratic People's Republic of Korea (DPRK) declared itself withdrawn from the treaty in 2003. However, there is disagreement among the remaining States as to whether the North Korean government met the treaty's conditions to have withdrawn successfully. Figure 2.1 provides a comprehensive global summary of every State's current NPT status.

Concerning safeguards, Articles I, II, and III of the NPT are the most relevant. The first two articles are complementary, applying respectively to the NWSs and NNWSs. Article I requires that NWSs not provide assistance in any way to facilitate the acquisition of a nuclear explosive device by a NNWS [15]. Likewise, Article II expresses the equivalent mandate for NNWSs, stipulating that they refrain from pursuing the acquisition of nuclear weapons. Then, building on Article II, Article III requires that each NNWS accept IAEA safeguards on all of its source or special fissionable material [15]. These safeguards must be delineated in an agreement concluded promptly between a NNWS and the IAEA. As mentioned previously, Article III substantially bolsters the IAEA's authority as described in its statute, compelling NNWS treaty signatories to permit the agency to administer safeguards on all of its nuclear material. With so many States party to the NPT, the IAEA is able to safeguard essentially all of the world's nuclear material not under the control of a NWS (or a non-party).

[3] Ireland sponsored the 1961 resolution, along with three previous resolutions in 1958, 1959, and 1960. Only the 1961 resolution passed unanimously [16].

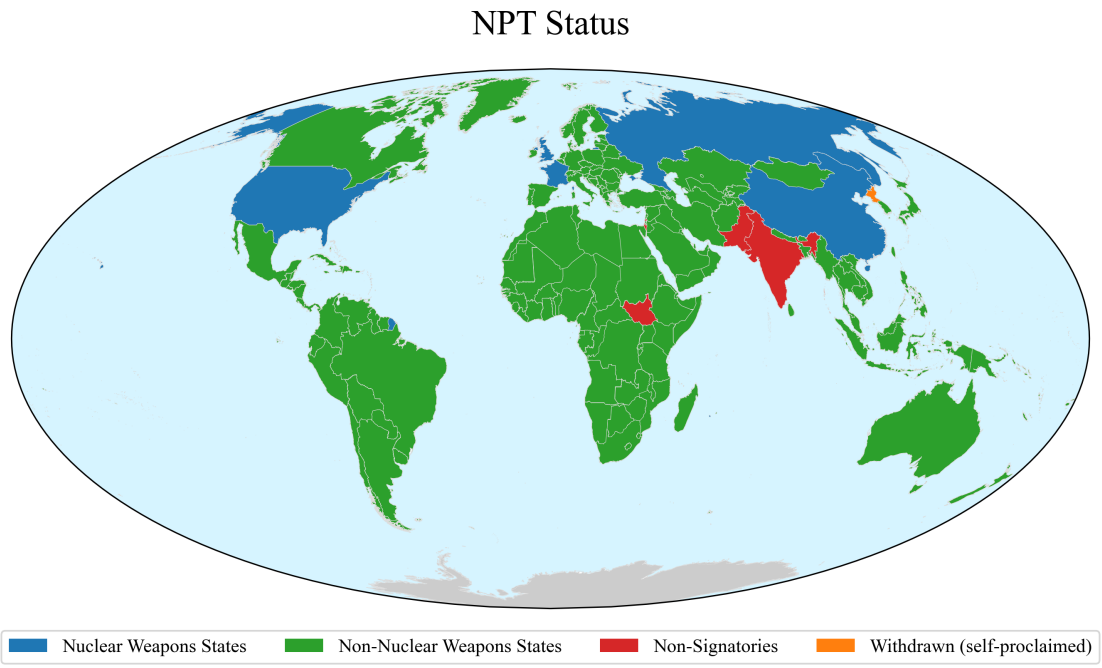


Figure 2.1: Global participation in the NPT [18].

Despite its prominence, the NPT is not the only international agreement regarding nuclear safeguards. Safeguards are also considered and required by other international treaties and agreements reinforcing nuclear nonproliferation commitments outside of the NPT. Relevant treaties include the Treaty for the Prohibition of Nuclear Weapons in Latin America and the Caribbean (Treaty of Tlatelolco) [19] and the South Pacific Nuclear Free Zone Treaty (Treaty of Rarotonga) [20], the African Nuclear Weapon Free Zone Treaty (Treaty of Pelindaba) [21], as well as similar agreements in Central and Southeast Asia [22, 23]. All of these treaties obligate the participating parties to conclude safeguards agreements with the IAEA.

International treaties do not all rely on the IAEA to enact safeguards, however. As an example, Brazil and Argentina jointly agreed to a bilateral agreement in 1991 to make safeguards consistent between the two adjacent countries [24]. That agreement creates an independent organization to manage the systems controlling nuclear material between the two nations, the Brazilian-Argentine Agency for Accounting and Control of Nuclear Materials (ABACC). While the agreement exists separate from any safeguards agreements that the two States conclude with the IAEA, it does encourage the IAEA and the ABACC to work together, sharing resources and avoiding duplicate safeguards efforts.

2.1.2 State Agreements

As stipulated by the NPT, NNWSs are required to abide by agreements that they conclude with the IAEA permitting the regulator to administer safeguards on all nuclear material belonging to the State. These agreements are contemporarily known as comprehensive safeguards agreements (CSAs)[4] and provide definitions and direction for the States to cooperate with the IAEA. Unlike the NNWSs, NWSs are not required to conclude CSAs with the IAEA; however, all five NWSs party to the NPT have elected to follow voluntary offer agreements (VOAs) that provide an analogous framework.

Though each CSA is case-specific and unique to a State, they generally follow a similar structure, outlined by the IAEA in INFCIRC/153 [13]. For example, each State must establish a *State or Regional Authority with responsibility for safeguards* (SRA) to bear responsibility for nuclear material accountability, along with a *State system of accounting and for and control of nuclear material* (SSAC) through which material accounting is performed and the IAEA applies safeguards [26, 27]. Additional protocols may also be concluded between States and the IAEA to expand the scope of safeguards beyond what is traditionally covered by comprehensive safeguards agreements.

2.1.3 Domestic Safeguards

International regulations are not the only form of nuclear safeguards that are in place worldwide. In fact, even in NWS where civilian nuclear operations are only subject to IAEA safeguards voluntarily, similar systems may exist between a national regulator and independent nuclear facilities. The United States serves as one example of a NWS that enforces its own national safeguards initiatives independent of the IAEA. The NRC imposes its own domestic safeguards program of nuclear material control and accounting (MC&A) to prevent the misuse of special nuclear material (SNM), along with other security measures. Like international safeguards, these MC&A practices work to ensure that nuclear facilities are operating as expected and that nuclear material is not stolen, lost, or otherwise diverted [28]. Similarly, the United Kingdom's Office for Nuclear Regulation acts in a comparable role, facilitating IAEA safeguards established via its VOA.

Although the remainder of this work will focus specifically on international safeguards, it is worth noting here that many of the dynamics occuring in the realm of international regulation are reflected in national relationships as well—now occuring between a nuclear facility and the national regulator as opposed to between a State (and its nuclear facilities) and the IAEA.

2.2 Safeguards Considerations

While the IAEA statute gives the agency broad latitude to prevent the diversion of nuclear material for any form of misuse,[5] the chief concern of the IAEA and its members is the nonproliferation of nuclear weapons [13]. From that perspective, any successful safeguards implementation must

[4] CSAs were formerly known as "full scope agreements" [25].

[5] Illicit high or low yield nuclear explosives, dirty bombs and similar radiological devices, or any other minacious machinations of a malfeasant.

take into careful consideration the steps required by any actor—state or non-state—to construct a nuclear explosive device. Nuclear explosives use some methods and materials that are distinct from conventional explosives, and good safeguards account for and identify their signatures.

These unique distinctions of nuclear explosives are, in fact, a boon for safeguards administrators. While many of the finer details of nuclear weapon design are kept classified by NWSs, the specific nuclear material types and the approximate quantities necessary for developing a weapon are in the public domain. Just as importantly, although the specific design of a nuclear weapon may affect the SNM requirements to some degree, it does not appreciably affect the implementation of nuclear safeguards. Together, these two facts allow the IAEA, an international organization without intimate weapons design information, to make informed assessments about proliferation risks and to develop technologies and procedures to reassure its members against the proliferation of weaponizable nuclear material. At the same time, actually procuring sufficient SNM to manufacture a nuclear explosive is recognized to be one of the most challenging physical acquisitions for a weapons program, and so serves as an ideal point in the weapons development process for the IAEA to monitor against.

2.2.1 Nuclear Weapons Material

The fissile nuclei at the core of a nuclear explosive bestow the uniquely destructive capabilities of the device. When a critical mass of these fissile isotopes is assembled, a chain reaction of nuclear fission ensues that results in the release of vast quantities of energy. To provide insight about the sheer power of a fission weapon, it can be compared against a traditional chemical explosive. The National Institute of Standards and Technology (NIST) defines the energy equivalent release of 1 kg of TNT as 4.612 kJ, or just slightly more than 10 eV per TNT molecule [29]. A single fissioning nucleus releases approximately 200 MeV in energy [30], a factor of 20 million more powerful than the molecule of TNT.

Nuclear fission occurs only for heavy isotopes where the Coulomb repulsion between positively charged protons in the nucleus is competitive with the strong nuclear force binding all of the nucleons together. The most tightly bound nuclei—those with the greatest binding energy per nucleon—have approximately 60 nucleons, as shown by the peak in Figure 2.2. While isotopes with more nucleons than this may find it energetically favorable to split into fragments to be more tightly bound, enough energy must first be imparted to the original nucleus to allow it to overcome the strong Coulomb potential just outside of the range of the strong force.[6]

In practice, only a handful of actinide isotopes have a required activation energy (also frequently called the critical energy for fission) low enough to experience fission at all. Among those isotopes, it is standard practice to differentiate between ones that are *fissionable* and those that are *fissile* [26, 30]. Nuclei are considered fissionable if they will undergo fission after a modest critical energy threshold is met. Since the neutron capture process on a heavy nucleus will release binding energy, a fissionable nucleus typically requires that the captured neutron bring at least 1 additional MeV into the reaction to trigger fission. Fissile nuclei, on the other hand, are a subset of fissionable

[6]For a more complete discussion of the physics of nuclear fission, see Krane, Chapter 13 [32].

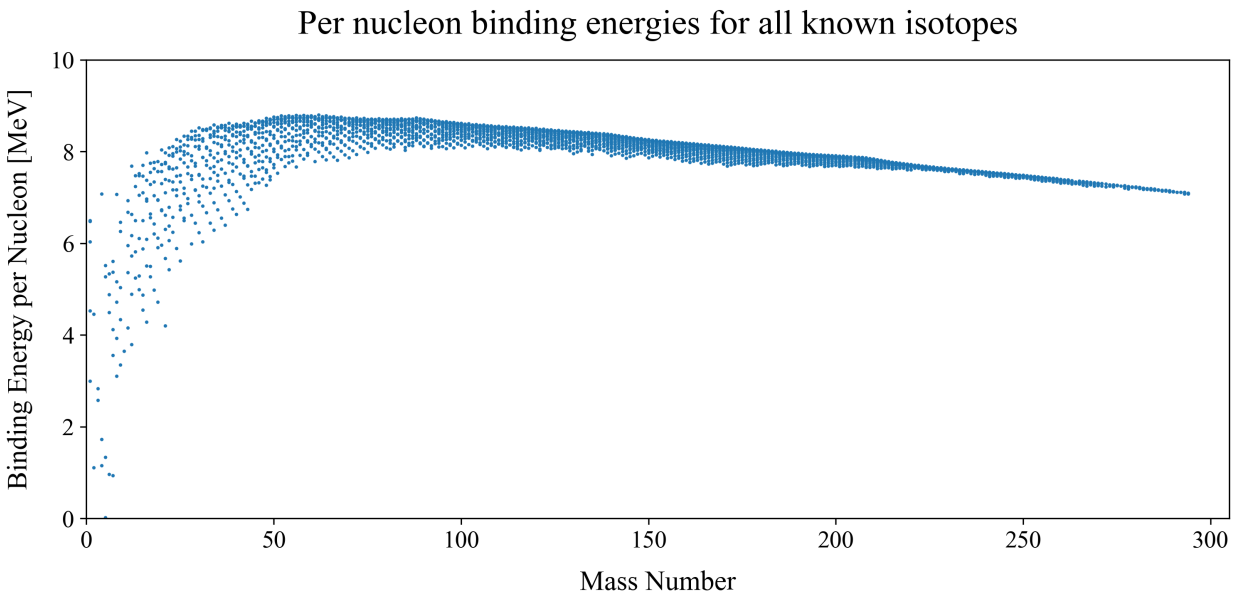


Figure 2.2: The binding energy of an atomic nucleus per nucleon, shown for all isotopes in the U.S. National Nuclear Data Center (NNDC) database [31].

nuclei where just the binding energy released in the capture process alone is enough to surmount the critical energy threshold and spur a fission reaction. Table 2.1 shows the critical energy for fission that is required by several fissionable isotopes, along with the energy released in a neutron capture event. The table demonstrates how fissile nuclei tend to be actinides with an odd number of neutrons.

2.2.2 Nuclear Weapons Physics

It is fissile nuclei that constitute the most essential component of nuclear weapons systems. For a fission chain reaction to be sustained, the product neutron(s) from each fission reaction must go on to trigger at least one additional fission event; in other words, the system must be critical. In a nuclear weapon, however, enormous amounts of energy need to be rapidly released. The majority of fission reactions driving the explosion must be complete before the explosive force disassembles the weapon's core, rendering it subcritical. This "explosion time" varies, but for common weapon designs the time between when a weapon core reaches criticality and when expansion returns the apparatus to a subcritical configuration tends to be on the order of a microsecond or less [34, 35].

To cause such a swift release of energy, the chain reaction must become supercritical, with the number of fission events increasing exponentially over time. Whether enough fission events occur for the system to reach supercriticality depends on a variety of factors: the number of neutrons produced during a fission reaction, the probability of any one of those neutrons subsequently in-

Table 2.1: A comparison of the binding energy released during a neutron capture event on a specific target nuclide [31, 33] and the critical energy required for fission [32], for a selection of fissionable isotopes. Fissile isotopes (in **bold**) are those with a value of released binding energy greater than the critical energy.

Target Nuclide	Protons	Neutrons	Neutron Capture Energy [MeV]	Critical Energy [MeV]
^{232}U	92	140	5.8	6.9
^{233}U	92	141	6.8	6.5
^{235}U	92	143	6.5	6.2
^{238}U	92	146	4.8	6.6
^{236}Np	93	143	6.6	5.9
^{237}Np	93	144	5.5	6.2
^{238}Np	93	145	6.2	6.0
^{239}Np	93	146	5.1	6.3
^{238}Pu	94	144	5.6	6.2
^{239}Pu	94	145	6.5	6.0
^{240}Pu	94	146	5.2	6.3
^{241}Pu	94	147	6.3	6.0

teracting with a fissionable nucleus, and finally, the probability of any such interaction resulting in fission. Given the last factor, it is at least intuitive that fissile nuclei are better suited for nuclear explosives than fissionable nuclei—the energy barrier preventing fission is lower.

Upon closer analysis, it turns out that fissile nuclei are the *only* realistic option for driving weapons-capable supercritical systems. While a nuclear weapon can include (and utilize) both fissile and fissionable nuclear material, it requires sufficient quantities of fissile material for the reaction to achieve supercriticality. Mixtures with enough fissile material to sustain criticality are not found naturally, and so material must be enriched to contain greater proportions of fissile nuclei to fissionable nuclei. A plot showing how the critical masses for spheres[7] of fissionable mixtures shrink as a function of enrichment in the fissile isotopes uranium-233 and uranium-235 is shown in Figure 2.3.

Since only fissile nuclides satisfy the requirements of a nuclear explosive to produce enough fission reactions before the weapon's explosive power disassembles the critical mass, IAEA safeguards are devised with fissile material as the primary focus. Safeguards objectives are defined in consideration of relevant fissile material quantities, and apply both to fissile material and other

[7] A sphere is the material geometry with the minimum surface area to volume ratio, and so neutrons are less likely to escape a spherical system without inducing another fission reaction than any other configuration. Therefore, spheres represent the minimum critical mass that can be achieved for any given mixture of fissile isotopes.

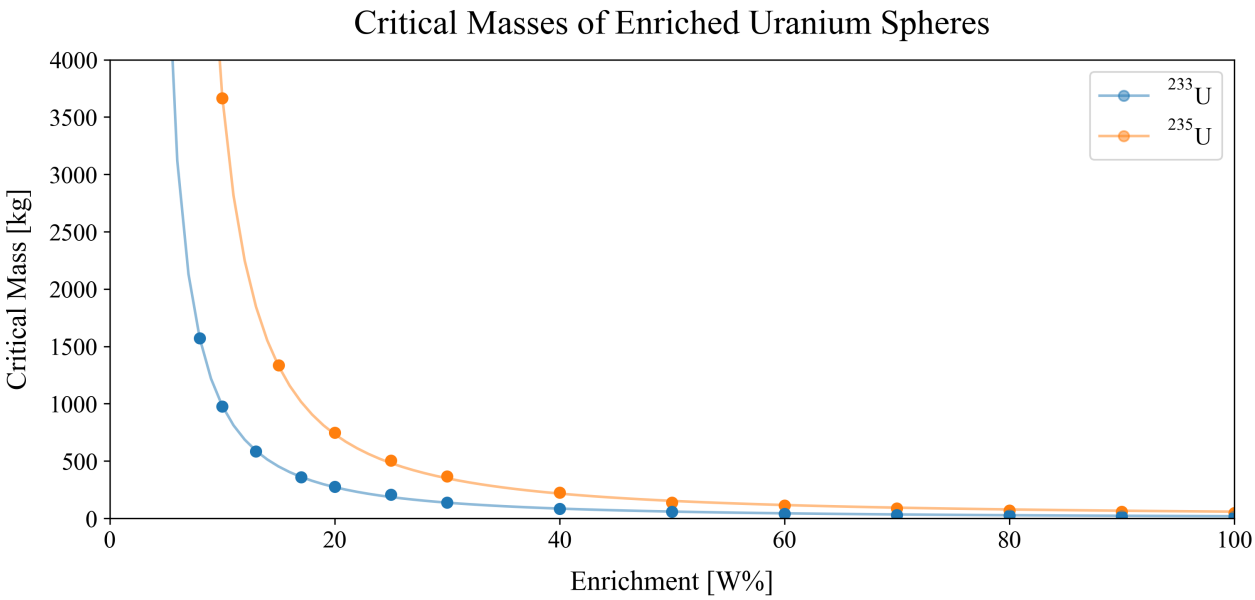


Figure 2.3: The critical mass of a bare metal sphere of a uranium mixture decreases as the mixture is enriched to have a greater fraction of either uranium-233 or uranium-235, both fissile isotopes [36: Adapted from Figure 3.1]. When minimal enrichment occurs, the size of the critical mass tends towards infinity, and so natural uranium (except in exceedingly large quantities) is not a proliferation risk.

source material from which fissile material may be derived.

2.3 Safeguards in Practice

From basic knowledge of nuclear physics and material properties, the IAEA sets appropriate safeguards goals to ensure that nuclear material is not being diverted. These IAEA inspection goals, specific to each facility under safeguards, include both a quantity component and a timeliness component.

The quantity component of the IAEA's inspection goal follows from the discussion in the previous section—that a critical mass of fissile nuclear material is required to detonate a nuclear explosive. Just as illustrated in Figure 2.3 for uranium, the critical mass of a mixture of fissionable isotopes is determined by the enrichment level of any fissile isotopes. Since this quantity of nuclear material required varies by isotopic mixture, the IAEA defines one significant quantity (SQ) of a material type as the approximate amount of material beyond which it is impossible to exclude the possibility of manufacturing a nuclear explosive [26].[8]

[8]The IAEA notes that SQs account for "unavoidable losses due to conversion and manufacturing processes" and are greater than what would be required to assemble a critical mass [26].

Table 2.2: Significant quantities of direct and indirect nuclear material [26].

Material	SQ
Direct use nuclear material	
Pu (^{238}Pu < 80%)	8 kg Pu
^{233}U	8 kg ^{233}U
HEU (^{235}U $\geq$ 20%)	25 kg ^{235}U
Indirect use nuclear material	
U (^{235}U < 20%)	75 kg ^{235}U
Th	20 t Th

For SQ definitions, the IAEA classifies materials into two broad categories: direct use or indirect use nuclear material. Direct use nuclear material is fissile isotopes and mixtures containing substantial fractions of fissile isotopes: highly enriched uranium (HEU), uranium-233, plutonium mixtures low in plutonium-238, or any similar mixture that could be readily used for weapons production. Indirect use nuclear material, on the other hand, is any other nuclear material that can be used to produce direct use nuclear material. Natural, low-enriched, and depleted uranium—along with thorium—are all considered to be indirect use nuclear material. A comprehensive list of SQ definitions are provided in Table 2.2 [26].

The second aspect of the IAEA's inspection goal is a timeliness component. Material diversions must be identified in a reasonable time, lest the detection time be too late for any meaningful actions to be taken. Additionally, by defining precise timeliness goals, the IAEA forms a quantitative basis for establishing inspection frequencies. Like the quantity component of the IAEA inspection goal, timeliness detection goals also depend on the material category. Higher risk materials are subject to tighter detection time goals than lower risk materials. For example, in the case where no other detection goals are agreed upon between a State and the IAEA, the standard detection goals are the following: one month for unirradiated direct use nuclear material (material requiring minimal processing before weaponizable); three months for irradiated direct use nuclear material (material requiring some processing before weaponizable); and one year for indirect use nuclear material (material requiring significant processing before weaponizable) [26].

Separate from these two inspection goals, it was recognized that the IAEA's responsibility to guard against nuclear material misuse and weaponization went beyond simply guaranteeing *non-diversion* of previously declared material, but also extended to monitoring against the possibility of weaponization of undeclared material. This understanding emerged from IAEA experiences in the early 1990s, when safeguards systems focusing on declared material were determined ill-suited for identifying undeclared nuclear activities, such as those existing in Iraq or the DPRK. To this end, the IAEA developed the Model Additional Protocol (INFCIRC/540) to extend the traditional

requirements of CSAs to cover undeclared nuclear endeavors [37].

These additional protocols task the IAEA with ensuring that safeguards are comprehensive and complete—that not only is nuclear material not being diverted, but that there is no nuclear material existing within a State that is not either subject to safeguards or exempt. Accomplishing this larger objective is nontrivial, requiring safeguards that rely more heavily on sophisticated methods of surveillance and monitoring analytics, design information for nuclear facilities to preclude the existence of undeclared nuclear activities at a site otherwise submitting to safeguards, as well as increased access for inspections [27]. They must also take into consideration potential acquisition strategies that a State might use for procuring nuclear weapons materials [26].

These goal sets—both the quantity and timeliness goals for detecting the diversion of previously declared nuclear material, as well as the IAEA's goal to confirm the non-existence of undeclared nuclear activity—are grouped by the IAEA into a set of three generic safeguards objectives. These objectives cover the complete set of scenarios in which a State may be engaging in nuclear material misuse. They are, briefly, to focus on detecting declared nuclear material diversions, then detecting undeclared nuclear activity at declared facilities, and finally detecting undeclared nuclear activity anywhere in a State [27]. These three generic objectives chart the course for international regulation.

2.3.1 Safeguards Modalities

While the generic safeguards objectives give the IAEA purpose, they are translated into technical objectives for implementation. These technical objectives can vary for any given State and are determined by its nuclear activities and capabilities, devised to exist within the framework established by the State's CSA. Appropriate and adequate safeguards are then deployed to satisfy these technical objectives. To actually coordinate a consistent and robust safeguards regime, the IAEA synthesizes a wide variety of information types to draw high-confidence conclusions that nuclear material is not being misused. Types of information that are brought together include information about facility design, regular and verifiable reporting of nuclear material inventories and activities, and the results of routine inspections.

First, for every nuclear facility under safeguards, design information must be provided to the IAEA and kept current. This information includes a description of the facility, the facility's layout, and how nuclear material flows through the facility [26]. The design information is used by the IAEA in developing a safeguards approach, applying safeguards appropriately to the specific facility, and ascertaining a State's compliance with any additional protocols [13]. For facilities that exist prior to the conclusion of a CSA, design information must be promptly submitted to the IAEA after the agreement is formally concluded [27]. For new facilities, or for alterations in existing facilities, the IAEA must be notified so that it may add or adjust safeguards as necessary. To prevent States from submitting false design information, and to discourage facilities from being either repurposed or modified to facilitate illicit nuclear operations, the IAEA retains the authority to inspect and verify all designs throughout the entire lifetime of the facility, including during both construction and decommissioning [13, 27, 37].

Material accounting represents a second major focus area of safeguards. To facilitate accounting, the IAEA designates material balance areas (MBAs)—physical areas where the quantity of nuclear

material can be accurately determined. Under the guidelines set forth in each State's CSA, design information provided by facilities may be used to delineate the boundaries of each MBA [13: Paragraph 46(b)]. Within an MBA, the IAEA may establish key measurement points (KMPs) to streamline and secure the safeguards process, improving efficiency while limiting discrepancies [13: Paragraph 108]. For example, in an MBA defined by a single building, KMPs may be located at entries and exits so that inputs and outputs to the MBA are properly identified. Alternatively a KMP could also be located in a common storage location for nuclear material where quantities of material should remain constant or predictable. With these types of systems in place, nuclear material does not need to be continuously tracked within an MBA because it can be considered to be safeguarded at all times.[9]

Beyond establishing MBAs, material entering or exiting an MBA must be recorded and reported to the IAEA along with an accounting of any other changes to the balance of nuclear material in the area. These additional adjustments include changes in form, such as material enrichment, transmutation, or burnup [38]. Once tallied, material balance records must be submitted regularly on material balance reports (MBRs) to the IAEA so that it may maintain an up-to-date inventory of the nuclear material under safeguards. Besides the known inventory listings and inventory changes, these MBRs include information on rounding adjustments to measured quantities and material that may be otherwise unaccounted [13: Paragraph 67].

Inspections constitute a third essential component of the IAEA's safeguards regime. By conducting regular inspections, the IAEA can independently confirm that a facility is operating as expected and complying with all other applicable safeguards. While inspections traditionally focus on facilities and areas defined by the State's CSA, States that have concluded an additional protocol with the IAEA must allow the IAEA to also inspect other locations of interest [37: Article 5]. In all cases, State agreements require that the State facilitate quick and easy access for IAEA to conduct inspections as necessary. This includes providing inspectors with adequate access to the facilities being inspected in a timely manner, as well as providing inspectors with necessary documentation and visas to legally enter the State [13: Paragraph 86, 37: Article 12].

IAEA inspections may be either announced or unannounced, but follow directly from guidelines set forth in a State's CSA. While unannounced inspections are a far stronger deterrent against misbehavior, it is recognized that unannounced inspections are likely to be significantly more disruptive to facility operations. As such, the IAEA provides facilities with relative frequencies for unannounced inspections along with a "general programme" of inspections to reduce the difficulties of accommodating unannounced inspections [27]. Still, under a State's applicable CSA, a facility may still be required to participate in short notice random inspections (SNRIs) if the IAEA elects to perform such an inspection [39]. In the SNRI process, a facility makes frequent, regular declarations to the IAEA regarding measured quantities of interest and its own records. These "mailbox declarations" are securely and immediately transmitted to the IAEA, while the facility is required to keep the material subjects of the declaration available for a predetermined "residence time". If the IAEA initiates a random inspection, the residence period guarantees that they may arrive in time to

[9]While not a foolproof system, ideally the KMPs are set up so that a party removing material from an MBA without passing through a KMP (e.g. exiting a building through a window) would arouse sufficient suspicion to be deterred.

successfully verify the integrity of the declaration against the facility's physical inventory.

2.3.2 Safeguards Tools

The IAEA uses a wide variety of specialized tools to successfully implement safeguards that satisfy its technical objectives. These include a myriad of radiation detectors, unattended remote monitoring systems, containment and surveillance devices, and strong data security mechanisms. While some of these tools are industry standards, others are highly tailored adaptations designed to meet the unique needs of a nuclear safeguards administrator.

Destructive and Non-destructive Analysis

For nuclear inspectors and regulators, possessing the ability to characterize nuclear material that enters, exits, or is stored in an MBA is of primary importance. Accurately assessing the quantities of nuclear material present in a facility or location is essential to successfully complete nuclear material accountancy technical objectives. In this regard, high-resolution gamma-ray detectors are arguably the most useful tools available to inspectors. Material isotopics can be determined using gamma-ray spectroscopy, and material quantities can be extrapolated from the gamma-ray intensity emitted by a given sample.

Most radioactive isotopes, including fissile isotopes that are of interest to safeguards practitioners, can be identified specifically by their unique gamma-ray emissions. For example, fissile uranium-235 alpha decays with a half life of more than 700 million years, but each decay has a 57% chance of emitting a 185.7 keV gamma ray when the product thorium-231 isotope de-excites. Detecting a strong 185.7 keV signal in a measured spectrum, like that shown in Figure 2.4, indicates the presence of a significant quantity of uranium-235. Likewise, mixtures of plutonium also produce distinct gamma-ray spectra depending on the isotopic quantities and so can be identified and characterized using gamma-ray spectroscopy.

In order to measure spectra with enough resolution to draw conclusions, the IAEA relies primarily on gamma-ray detectors that use high-purity germanium detector (HPGe) scintillators. HPGe detectors are the industry standard for generating high-resolution gamma-ray spectra. Although the high-purity germanium crystal composing the scintillator must be kept cryogenically cooled, electrically-cooled portable detectors are used by inspectors to monitor facilities. One such handheld gamma-ray spectrometer is shown in Figure 2.5a.

In addition to material characterization using gamma-ray spectroscopy, neutron detectors may also be used by inspectors to measure nuclear material of interest. These detectors are highly valuable when seeking to identify the presence of SNM, as that material has the potential to emit neutrons either spontaneously or via an induced reaction. First, passive neutron detectors may be able to detect the presence of neutron emitting isotopes in the vicinity of the detector system, such as those that undergo spontaneous fission. Spontaneous fission is a decay mode accessible to fissionable nuclei with large mass numbers, and the process is often a strong source of neutrons. Mixtures of plutonium, for example, are prone to spontaneously fission making them well-suited for detection via neutron monitoring. Another alternative to these passive neutron detectors are active methods

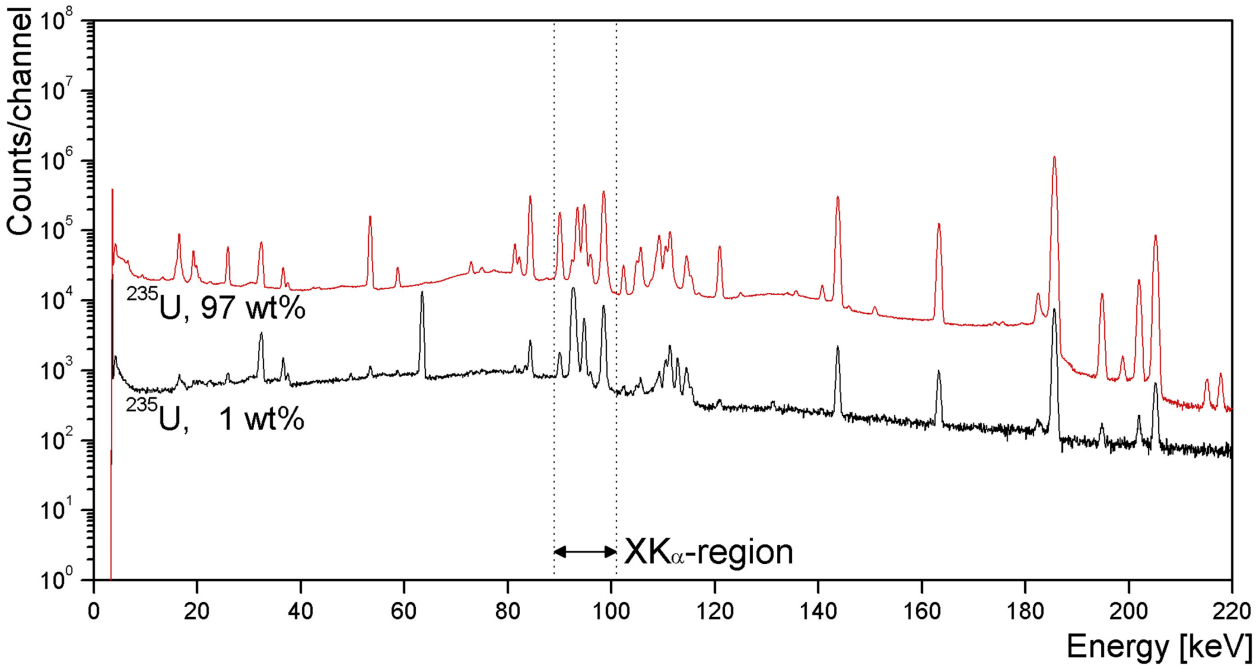


Figure 2.4: Measured gamma-ray spectra for two samples of enriched uranium-235 (at 1% and 97% by weight), as reported by Choi and Kim [40: Figure 1]. The discernible peak at 185.7 keV is produced by uranium-235 decaying into thorium-231. (Although the XK$_\alpha$ region highlighted by Choi and Kim is related to the behavior of the atom after emission of the 185.7 keV gamma ray, it is of little use to nuclear inspectors and can be ignored here.)

of neutron detection, where neutron interrogation is used to induce fission reactions. In these systems, a source of neutrons–for example, an americium-lithium (AmLi) neutron source—induces fission in the nuclei of SNM, with the resulting neutrons being measured by the detector system. Figure 2.5b shows a well-style neutron interrogation detector.[10] In either case, once material isotopics are either known (e.g. from gamma-ray spectroscopy) or surmised, neutron counters can also provide an effective tool for gauging the quantity of material present in that location.

Both gamma-ray spectroscopy and neutron counting are considered non-destructive analysis (NDA) techniques and are useful for characterizing material without affecting the sample. In general, NDA techniques may be preferable for both the safeguarded facilities and the IAEA. From the perspective of a nuclear facility, NDA techniques may be less intrusive, requiring no loss of material or extraction of samples [43]. For regulators, NDA techniques tend to be more cost effective than alternative destructive analysis (DA) methods, which require expensive laboratories and equipment [42, 43]. Benefiting both parties, NDA measurements are often repeatable, and can

[10]The well-style is named since a small sample of material may be placed inside the "well" of the detector. Other styles are better suited for larger objects, such as the "collar" style used for inspecting reactor fuel assemblies.

(a) (b)

Figure 2.5: IAEA radiation detection equipment: (2.5a) a handheld gamma-ray spectrometer connected to a laptop computer displaying the measured spectrum from a sample [41]; and (2.5b) a well-style neutron interrogation detector [42: Figure 9].

be reproduced in the event of a dispute [43]. Despite those advantages, DA techniques still serve a distinct role in performing high-precision analyses. Measurements using mass spectrometry or chemical titration may provide highly accurate information about the composition of a material sample, allowing an regulators to discover bias defects in material quantities over long time periods [42]. Even though they can be highly accurate, DA assessments prevent further use of the material and are more expensive, and so are generally relied on in a more limited way than NDA, often for the calibration and verification of instrumentation and procedures [42].

As technological advancements have improved the quality of NDA insights, the IAEA has begun placing greater emphasis on unattended monitoring systems (UMSs) in its safeguards portfolio. These systems include a selection of NDA tools that are designed to operate remotely and without full-time supervision [42]. These devices may record data to a secure local filesystem for future download by a credentialed inspector, or they may transmit data securely to an external location—for example databases at the IAEA headquarters.

The UMSs used by the IAEA exist in a variety of types. One device, the on-line enrichment monitor (OLEM), is installed in enrichment facility piping to determine the throughput of enriched material (uranium-235) [44, 45]. The device collects temperature and pressure readings, which permit calculation of the gas density of the flowing uranium hexafluoride. Simultaneous spectroscopic readings from a mounted sodium iodide detector then enable the determination of the uranium-235 enrichment of the pipe's contents. These measurements may then be used to assess compliance with enrichment limitations.

Another tool used for unattended remote monitoring is the unattended fuel flow monitor (UFFM).

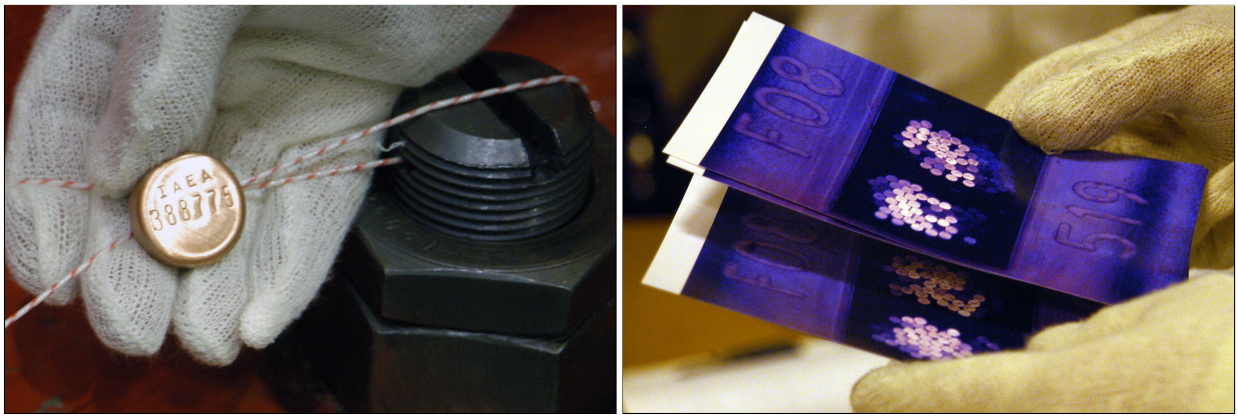
The UFFM is designed to facilitate monitoring of fuel assemblies—either fresh or irradiated—as they move through a facility. For irradiated fuel especially, substantial amounts of shielding are required to protect against the highly radioactive fuel assemblies. This shielding makes the inspection process more challenging, as the irradiated fuel assemblies may not be visually counted [46]. Furthermore, reactor refueling periods may last months, and it becomes impractical for an inspector to remain at the facility for the duration of the refueling cycle [46]. Instead, the UFFM can be installed unobtrusively in a key location, such as the door-valve to a reactor pressure vessel, where it can both detect the (directional) motion of fuel assemblies (and other objects) past the detector and characterize their irradiation levels [42, 46].

In a third UMS variation, technologies like the plutonium inventory monitoring system (PIMS) also enable comprehensive remote monitoring implementations to satisfy safeguards requirements. The PIMS is a sensor system that continuously records data regarding the quantity of plutonium in a plutonium powder process area [42]. The system employs 142 neutron counters strategically placed throughout an area to measure plutonium quantities. Where the system surveys material presence continually, regulators may receive immediate notice if material is unexpectedly removed. Unlike the OLEM and UFFM systems, which measure the motion of nuclear material past the unattended detector, tools like the PIMS track stationary material inventory that is not expected to change regularly (or unpredictably).

Containment and Surveillance

So far, only safeguards systems that explicitly enable nuclear material accountancy have been covered. However, the IAEA also relies on containment and surveillance technologies to complement their suite of material accountancy technologies, providing stronger guarantees against material diversion and misuse. Some of these technologies are relatively unique to the safeguards environment.

For example, tamper indicating seals are used to guarantee that sealed containers of nuclear material have not been opened. The most common of these are metallic cap seals [42], shown in Figure 2.6a. These single-use seals are uniquely numbered, and are only to be removed by safeguards inspectors. After removal, the seals are returned to IAEA headquarters for inspection of the caps soldering and general wear to guarantee their authenticity. More recently, the IAEA has begun to make use of more sophisticated sealing devices, with two notable technologies being the fiber optic general purpose seal (FBOS) and the electronic optical sealing system (EOSS), both of which are loop-style seals offering assurances against tampering. In the case of the FBOS, a digital imprint of the sealed fiber optic wire pattern is saved when the seal is first closed [42]. Then, any future attempts to break the fiber optic wire or detach and reattach the seal will almost certainly reconfigure the fiber optic pattern. Upon future reinspection, any fiber optic wire configuration other than the original suggests that a party tampered with the seal. Figure 2.6b gives an example of the images regulators use to compare these patterns. Other sealing technologies, like the EOSS employ more active monitoring methods. The EOSS transmits light signals through a fiber optic loop (similar to the FBOS) but at short sub-second intervals. If the seal is opened, the light pulses no



(a) (b)

Figure 2.6: IAEA sealing equipment: (2.6a) a metallic cap seal used to secure safeguards equipment and material [47]; (2.6b) comparison images of wire imprints used in a fiber optic sealing system [48]—discrepancies indicate tampering.

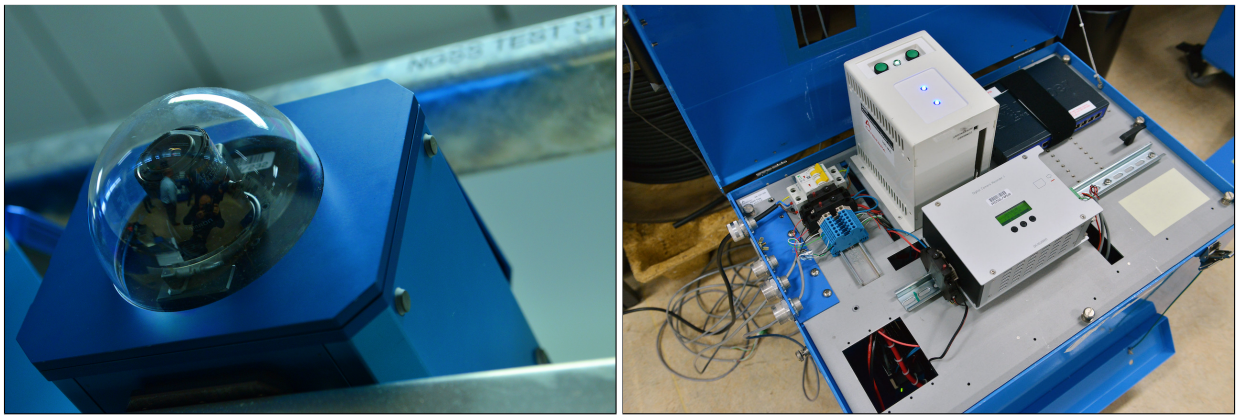
longer complete their trajectory through the cable, and the exact time and duration of the unsealing event can be recorded [42].

Other containment and surveillance technologies are relatively traditional but with unique characteristics. For example, surveillance camera devices like the next generation of surveillance system (NGSS) accomplish objectives similar to standard security cameras, but with customizations appropriate for use in a safeguards environment [42]. These cameras are placed in strategic locations either singly or in coordinated arrays and take periodic snapshots of facility activity. Unlike conventional security cameras, however, these cameras generally have lower frame rates (capturing images at intervals of one second or longer). They are also housed in a tamper indicating enclosure, the blue camera casing shown in Figure 2.7. Like some of the unattended detector systems, data may be stored locally or transmitted to a remote server [42]. Either way, the system is designed to prioritize integrity, enabling the IAEA to trust that the monitors are relaying accurate and trustworthy safeguards information.

Data Authenticity

Especially as UMS systems become more prevalent, the amount of digital data acquired by and accessible to the IAEA is growing rapidly. At the same time, all data collected by the IAEA must be protected and secured to guarantee that those conclusions are accurate. If the data becomes compromised, the IAEA will be unable to guarantee the integrity of its conclusions and that could weaken the trust that it builds with member States. To preserve the integrity of the data—and to protect the potentially sensitive information that may be collected—the IAEA must also stay up-to-date on modern data security practices.

One way that the IAEA ensures the veracity of its data is procedural: the IAEA limits what is



(a) (b)

Figure 2.7: IAEA surveillance equipment: (2.7a) a next generation of surveillance system (NGSS) camera [49]; and (2.7b) an upgraded NGSS server [50]. Both systems are housed inside tamper indicating blue enclosures.

shared between the agency and the facility or State. In general, IAEA monitoring equipment is owned and operated by the agency, and the data is not otherwise shared [42]. There are exceptions to this rule, and certain devices are classified as joint use equipment (JUE). For example, the PIMS devices mentioned in Section 2.3.2 are owned by the facility and data is shared with regulators under the provisions of a joint use agreement (JUA) [42, 51]. However, most equipment and operating data are maintained and collected independently by the IAEA without the expectation that this information will be shared back to the State or facility. Naturally, this makes it less likely that a State or facility may exploit weaknesses in a system. Unfortunately, this policy may also limit the IAEA from leveraging all of the operational data that may be at its disposal. Valid, useful data that has been collected by the facility on equipment it owns may not be able to be trusted or used in supporting or denying safeguards conclusions. The policy also has the potential to place additional burdens on a facility that must support these independently operated systems, including costs of installation, maintenance, and inspection [51].

To enhance the integrity of facility-generated data and reduce the need for redundant systems, tools have been developed to enable increased cooperation. One such device, the enhanced data authentication system (EDAS), uses a branching system to split an input into two components. The first output branch is immediately duplicated and encrypted, with this cryptographically secure copy able to be transferred to a secure, trusted location. The second output branch produces a plaintext version of the detector signal, which may be routed back to the facility for operational use [52]. Ideally, the EDAS is installed as near to the data source (i.e. the detector) as possible, to reduce the potential avenues for data to be altered before reaching the device. In this way, a regulator may be able to infer significantly greater credibility to a dataset than if they were consulting the same data stored by a facility.

Besides rules limiting shared information between an inspector and the facility, the IAEA also

uses standard, secure encryption procedures to transfer electronic data and communications. Data that is collected at a facility by an UMS will be timestamped, encrypted, and transferred to a secure location—either a trustworthy local filesystem or a remote location [42]. Since the IAEA is obligated to protect the confidentiality of State information [37: Article 14(b), 15], the IAEA uses modern encryption schemes, public-private key authentication, two-factor authentication, and strictly permissioned systems.

In addition to securing data that it collects, communications between the State or facility and the IAEA may be initiated using a secure, online mailbox system. The system enables a facility to securely transmit and receive messages and data, guaranteeing that after submission the communications and records may not be altered. These properties make the mailbox system suitable for facilitating regulator actions like SNRIs [39]. When a State is given short notice of an inspection, they may also be required to furnish the IAEA with certain types of information that can be verified upon arrival of an inspector. Ideally, the difficulty of providing such a "mailbox declaration" that conceals misconduct—while simultaneously being consistent with the findings of inspectors and historical records—would be a severe deterrent against impropriety.

2.4 Challenges of Safeguards Cooperation

The IAEA's safeguards practices enable them to regulate the international civilian nuclear community thoroughly, setting a course for meeting their objective of preventing the misuse of nuclear material. Still, potential adversaries are constantly adapting and advancing, and so the IAEA must also be perpetually looking to strengthen its safeguards practices. For the most part, these improvements in safeguards systems on the part of the IAEA are not controversial, as they simply improve existing technologies that are covered by safeguards agreements already in place. More challenging is incorporating new modes and types of data, increased levels of monitoring, or emerging technologies into these safeguards systems.

2.4.1 Competing Interests

Building cooperation among international parties is no easy feat. While global organizations like the IAEA are conferred authority under international law (such as the NPT), States are often reluctant to yield sovereignty or accept restrictions from external parties. Fortunately, the value of nuclear safeguards is recognized with near universality; there is strong international commitment to the NPT and commitment to its principles even among non-signatories. Despite that common agreement, differences still exist regarding the exact interpretations of the IAEA's responsibilities and authority. Since the mission of the IAEA is to prevent the diversion of nuclear material, it is their responsibility to implement safeguards to the best of their ability, expanding their scope and advancing their technology as necessary to ensure their goals are achievable. At the same time, States have to protect their own best interests, including the protection of sensitive information, whether that pertains to matters of national security or proprietary business developments.

These competing interests present a challenge for safeguards negotiators. The IAEA seeks to verify nuclear compliance using the best data, algorithms, and technology at its disposal. It must execute this mission using trusted sources—often equipment that is operated independently of the facility operators to produce authentic (uncompromised) data. States may push back, limiting what they share with investigators because they are concerned that acquiescing to increased surveillance burdens them with more risk (even if they are operating with good intentions). This risk could arise from the potential exposure of sensitive security or proprietary information in the event that the IAEA's communication or data storage resources are compromised, even when that data is securely encrypted; such events might be caused by insider threats or cyberattacks. It might also be due to States' reluctance to trust the strength of those encryption methods, especially ones that they feel are more vulnerable to exploitation [25]. Alternatively, some States may be wary that international superpowers could attempt to leverage their power in the IAEA to exert influence on the agency's actions. States may also be resistant to accepting new surveillance that might add to the operational costs of a facility, either through the installation of physical hardware or by diverting labor resources to accomodate IAEA inspectors as they verify the authenticity of safeguards equipment [51].

It is in this niche that emerging technologies—such as privacy-preserving algorithms—may prove highly beneficial. Privacy-preserving computation, which is discussed in depth in Chapter 3, would provide the IAEA and State nuclear facilities with the ability to engage in a single joint calculation that would *guarantee* the privacy of the information supplied to the calculation. The cryptographic fundamentals underneath these privacy-preserving techniques assure that any inputs are never actually shared, and so they could therefore not be exposed in the event that IAEA communications or storage systems are compromised. Due to the underlying algorithms, some types of privacy-preserving computation may also even be accepted by States that are reluctant to trust other varieties of encryption. At the same time, the IAEA may be able to limit the amount of information that is shared back to a facility regarding its analysis techniques, preserving a degree of independence in its analysis for cases where systems like the EDAS increase the trustworthiness of input data streams from facility-owned measuring equipment.

Ultimately, privacy-preserving technology would bridge the gap between the wishes of regulators and facilities. It would enable the IAEA to derive insights from datasets that may have been previously inaccessible under conventional agreements and expectations, while assuring a facility that they're yielding minimal privacy through their participation.

2.4.2 Defining a Safeguards Threat Model

To move forward in presenting a solution that both protects the privacy of a State while simultaneously improving the safeguards conclusions drawn by the IAEA, it is instructive to consider the roles and assets of each party. With that perspective, a threat model can be developed to determine the characteristics required by a desired technology or technological system. Threat models of this nature are especially important when considering potential privacy-preserving algorithms, as they rely on cryptographic protocols that behave only under a specific set of assumptions. Outlining a threat model can help ensure that the assumptions are satisfied.

A safeguards system can be considered with just the two parties—the IAEA and a State or facility. Where the chief objective of the IAEA is to be able to detect the diversion of one SQ of nuclear material in some period of time, it follows that the goal of an "attacker" (or any party attempting to elude or undermine safeguards) is therefore to successfully divert at least one SQ of nuclear material in a short period of time without detection. Since safeguards administrators must assume that the State facility (or another agent of the State) may adopt this anatagonistic role and be actively attempting to divert nuclear material, the threat model dictates that any real solution be a system that is secure against malicious adversaries.[11] These are parties who may be attempting to cheat the safeguards protocol.

All safeguards systems must operate within this model, and two examples from the physical world illustrate the concept well. First, although the safeguards inspector may not know what is taking place at a facility at all times, random inspections (and especially SNRI) are designed to expose a party who may be attempting to evade their safeguards reporting requirements. Without knowledge of the inspection dates and times, it becomes much harder for a State to divert material without their action ever coming to the attention of the regulator. A second example is the IAEA policy that stipulates maintaining independence between regulator and facility equipment. In this case, the State or facility has access to measure any quantity available to it from the operation of the facility. The IAEA may have access to a subset of this information by installing its own independent monitors, which have their own intrinsic capabilities and limitations, as it is allowed under a CSA. As discussed in Section 2.3.2, by installing these systems independently and with various security mechanisms, the IAEA may trust that its readings and measurements are not being manipulated. It may also be more confident that the facility has not exploited knowledge of the IAEA's measurements (or analytical techniques) to undermine the agency's analysis.

In the event that the IAEA *does* use facility equipment, if the IAEA were able to verify the data collected by a facility instrument as accurate, they might be able to trust that data as a valid indicator of State compliance in their non-proliferation obligations. Emergent technologies such as the EDAS allow the IAEA to have greater confidence that readings from facility owned equipment have not been manipulated. By providing an encrypted data stream directly to the IAEA from a detector source, a malicious State would be unable to corrupt that output data. Such developments suggest a promising future for increased levels of data sharing data between the IAEA and States.

Considering this future, safeguards systems are constantly evolving. Section 2.4.1 emphasized how, to stay ahead of potential adversaries, the IAEA must always be seeking to build upon its ability to draw strong conclusions about a State's compliance with their non-proliferation requirements. To bolster confidence in its conclusions, the IAEA can either work to improve its current technological capabilities or extend its monitoring capacity. The former is already a focus for scientists and engineers developing IAEA technology and is not the subject of this dissertation. The latter is in some ways more challenging, requiring that States be convinced to share more information than they currently do under the status quo. For the IAEA, expanding monitoring capacity would also come at the expense of installing more independent measurement capabilities and UMSs, or performing more inspections.

[11] For a more detailed discussion of malicious adversaries in cryptographic protocols, see Section 3.4.

However, a situation could be conceived where the IAEA augments its current capabilities with non-traditional, highly sophisticated, remotely collected data streams—or even with high quality operating data collected by equipment owned by a facility as described before. These scenarios would presumably incur only a minimum additional cost to the facility. If such systems are realized—by either new types of data or leveraging facility systems—then the lone remaining challenge becomes convincing States and facilities to share the insights from their own measurements. While these entities will most likely be reluctant to share any more than the minimum amount of information required by their safeguards agreements to minimize risk, they may be more likely to cooperate if a digital system could be designed to grant the IAEA insight while simultaneously assuring the security of any inspected data. Such security guarantees would ensure that nothing beyond the mutually agreed upon insights would be exposed to anyone—neither third parties outside the interaction nor the IAEA.

This dissertation explores the creation of digital systems fitting this description using privacy-preserving computation. It is important to emphasize here that these digital, cryptography-based systems are generally independent of the physical systems used by safeguards administrators to perform monitoring of nuclear facilities. Those physical systems are often hardware devices, like many of the tools described in Section 2.3.2. This distinction is important, as even with computationally secure digital algorithms, the potential still exists that the underlying monitoring technologies could be exploited. In fact, in any real implementation of a privacy-preserving technique applied to the measurements of a physical safeguards system, a viable solution would need to be secure against malicious adversaries in the physical sense and the digital sense.

Physically, a real system relies on the fact that inputs faithfully represent reality; the inputs reflect data collected by an unencumbered[12] measurement device and are not spoofed, artificially generated values. Chapter 6 considers some potential solutions to these vulnerabilities—including how tools like the EDAS and commitment schemes may combine to be useful for verifying data integrity—but this dissertation does not present a final solution. While preventing such exploitation is a critical (and perpetual) area of research, protecting against those types of physical attacks is outside the focus of this work.

Digitally, a real system must guarantee that parties are executing the protocol as designed, and that they are not attempting to cheat by subverting the procedure. While this dissertation will initially assume cooperating, semi-honest parties in its demonstrations, it should be noted that any future real implementations ought to use algorithms with enhanced security protecting against hostile opponents. Fortunately, just as random inspections can introduce an unknown element into existing physical safeguards systems to deter adversarial behavior, similar techniques can help secure a semi-honest digital safeguards system against malicious adversaries. Section 3.4 provides some discussion of cryptographic protocols that would achieve this stronger security goal, and how they may be built from the simpler semi-honest constructions.

With that in mind, it is possible to devise digital safeguards systems that meet the goals of allowing a regulator to gain valuable insights into facility operation while simultaneously protecting the privacy of a facility's data. Ideally, in such systems it should be mathematically provable that

[12]Obstacles like strategically placed shielding could prevent a detector from collecting proper measurements.

the regulator learns only insights relevant to safeguards and nothing else; the privacy and data security of the State or facility in question will be entirely preserved. While a complete guarantee of security is certainly a strong claim, privacy-preserving computation offers a computational system with exactly this capability.

Chapter 3

Privacy-Preserving Computation

Privacy-preserving computation describes a class of algorithms that are performed between multiple parties while preserving the privacy of at least one of the participant's inputs to the calculation. In general, these algorithms all build on cryptographic foundations; however, they span a wide range of functionality. Fully homomorphic encryption (FHE) enables one party to perform operations on another party's encrypted data; zero-knowledge proofs (ZKPs) allow one party to prove that it knows some fact while another learns nothing other than the proof; secure multiparty computation (MPC) permits multiple parties to jointly compute some function of their combined inputs, while no party learns anything about any other party's input to the calculation.

There are a number of existing practical applications where flavors of privacy-preserving computation have already been incorporated successfully. These include online auctions, biometrics, cryptocurrencies, anonymous surveys, and contact tracing. In all of these instances, privacy-preserving computation has enabled two parties to share data and gain insights while being assured that their own information—whether that be bids, biometric identifiers, or average salaries by gender—would remain secure.

When considering the application of privacy-preserving computation to the challenges of nuclear safeguards, multiparty computation (MPC) specifically is a natural fit. Considering a computational program evaluated jointly by a safeguards regulator and a nuclear facility, MPC could allow the two parties to execute the protocol while neither must expose their private inputs. An example might be the joint evaluation of a calculation yielding some quantity of interest to the regulator that requires data from the nuclear facility beyond what the facility would be willing to share openly through a comprehensive safeguards agreement (CSA). Using MPC, the regulator could gain safeguards-relevant insights while the facility's data would be entirely protected. Taking this example further, since the International Atomic Energy Agency (IAEA) is often required to maintain a degree of independence from the facility operators, MPC also protects any evaluation criteria that they supply to a joint calculation based on a facility's measurements.

In such a scenario involving only two parties, calculations of this variety can make use of the subset of MPC protocols designed for the two-party case. Among these two-party computation (2PC) algorithms, garbled circuits emerged early on as a foundational technique in executing secure computations and have remained one of the most popular classes of algorithms for those tasks.

Because of this, garbled circuits were chosen as the primary instrument for building the privacy-preserving analysis and demonstration tools that will be presented in the following chapters. These are tools that will showcase how MPC can be applied to nuclear safeguards and facilitate the adoption of MPC by safeguards administrators. To lay the groundwork for those discussions, the case of two-party computation and garbled circuits in particular will be the central focus going forward.

3.1 Secure MPC and Garbled Circuits

The first MPC algorithms were discussed in the academic literature in the mid-1970s, and relied primarily on how to share secrets between multiple parties [53, 54]. Over time, these algorithms evolved [55], and in the 1980s Andrew Yao introduced the modern concept of MPC [56] followed by techniques to solve arbitrary problems using MPC techniques [57]. To frame the question he was trying to solve, Yao popularized the Millionaire's Problem, posed as the following:

> Two millionaires wish to know who is richer; however, they do not want to find out inadvertently any additional information about each other's wealth. How can they carry out such a conversation? [56]

In Yao's original oral presentations on the subject,[1] he introduced the general concept of using Boolean circuits as mechanisms for performing MPC [56, 57]. Yao's work was cited in work by Goldreich, Micali, and Wigderson, who are credited[2] with the first written description of circuit-based MPC [55]. In 1990, Beaver, Micali, and Rogaway proposed the name "garbled circuit" to refer to their construction built on a symmetric primitive—a modification of the Goldreich-Micali-Wigderson (GMW) technique [60]. Circuits that employ symmetric primitives in this way are contemporarily distinguished as "Yao's garbled circuits".

Secure MPC is defined as a procedure in which two or more parties jointly evaluate a function of their combined inputs, and where no party learns anything about the inputs of any other party besides what can be gleaned from the shared output of the function. Mathematically, this is formalized by defining a desired functionality $f : (\{0,1\}^*)^m \rightarrow (\{0,1\}^*)^m$, where $\{0,1\}^*$ denotes a string of 0s and 1s of arbitrary length, and m is the number of parties participating in the computation [61]. If $x_i \in \{0,1\}^*$ is the binary input of party $i \in \{1,...,m\}$, then $f(x_1,...,x_m)$ denotes the output of the functionality f with m distinct components and f_i denotes the output component accessible only to party i. In the two-party case, $m = 2$ and the functionality f is reduced to $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$.

In general, MPC protocols are expected to satisfy at least two fundamental assumptions: that they enforce *correctness* and *privacy* [62–64]. First and foremost, the calculation must guarantee that the answer be correct. If the algorithm does not yield the correct answer to the algorithm being evaluated, it is essentially useless. To guarantee correctness, the algorithm must output the answer

[1] Most publications cite Yao's articles in 1982 and 1986 as the origin of garbled circuits; Goldreich clarifies that the pertinent information was provided in the accompanying oral presentations [58: p. 194].

[2] Though the record is frequently offered in fragments, a relatively clear and complete history of garbled circuits is documented by Bellare, Hoang, and Rogaway [59].

that corresponds to the given inputs with a probability of exactly one. Second, a calculation must also preserve privacy, a premise that seems obvious enough. In a privacy-preserving calculation, nothing must be revealed about the inputs provided by either party other than any information that is directly conveyed by the result of the evaluated function. If privacy is not preserved then the algorithm is obviously not an instance of privacy-preserving computation. Formally, this means that the probability of producing the correct answer using anything less than a complete set of inputs is the same as the probability of producing any other potential answer from that (incomplete) set of inputs. More simply, any party that does not know all of the inputs to the privacy preserving protocol (which is, ideally, every party to the computation) ought to be no more likely to be able to deduce the correct answer from that information than if they were to randomly guess.

In addition to these two requirements, an MPC protocol may be expected to provide more rigorous security guarantees depending on the implementation scenario. Various security threat models exist, ranging from cases where parties are generally considered at least partially trustworthy, to cases where all parties to the computation must be treated as potential adversaries. In the former case, a notion of passive security is sufficient. Parties engaging with MPC in this way are deemed "honest, but curious" or semi-honest, and while they may be trusted to follow the protocol as it is defined, they may also attempt to use every piece of information that they receive during the course of the calculation to learn as much information as possible [63]. Despite being the least strict security models, passively secure protocols tend to be the easiest protocols to implement—both in terms of complexity and their computational burden. As such, passively secure protocols often form the foundation for building protocols with strong security assurances.

For cases where parties cannot be trusted at all, other more rigorous security models may serve as the basis for a privacy-preserving algorithm. In actively secure models, privacy is maintained even in the case that a party attempts to cheat or break the protocol [63]. This party may attempt to take some action that forces the circuit to produce the wrong answer (violating the *correctness* criteria), or alternatively, take some action that gives a public answer revealing one of the other parties inputs (violating the *privacy* criteria). Although such a malicious adversary may not be caught by an actively secure protocol, an honest party can be assured that their inputs to the calculation will be protected regardless. To deter malicious adversaries altogether, parties may turn to a covertly secure model where the protocol exposes a cheating party with some probability [63, 65]. While the ensuing descriptions of MPC will focus on passively secure systems, more discussion of protocols secure against malicious adversaries is offered in Section 3.4.

3.2 Yao's Garbled Circuits

Yao's garbled circuit protocol is one of the most popular techniques for implementing 2PC [63]. Like other circuit-based MPC techniques, the method is founded upon the fact that any computable function (i.e. any function that can be evaluated by a computer) can be formulated as a Boolean logic circuit. The circuit consists of individual wires—each holding a value of zero or one—connected by Boolean logic gates: `NOT`, `AND`, `OR`, `NAND`, `NOR`, `XOR`, or `XNOR`.

A sample logic circuit, a comparator, is shown in Figure 3.1. The comparator is a Boolean circuit

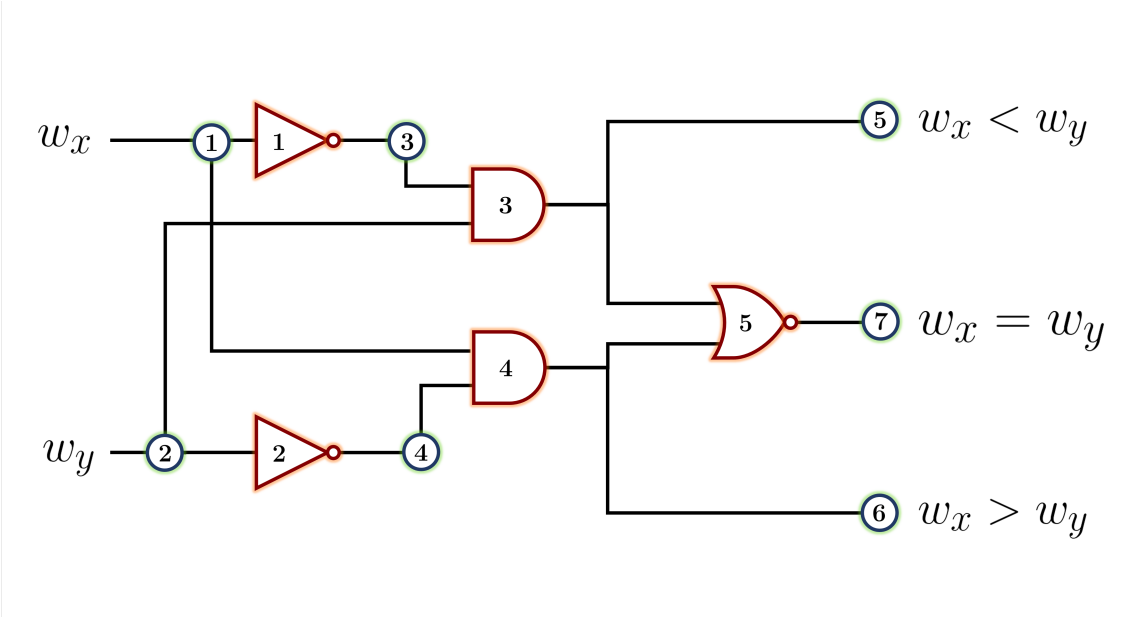


Figure 3.1: A comparator circuit evaluating whether the value on wire w_x is greater than, less than, or equal to the value on wire w_y. Gates are numbered in shapes corresponding to the IEEE/ANSI 91/91a-1991 standard and wires are numbered separately in circles. Only one of the three output wires will ever be true, indicating the result of the comparison.

representation of the function

$$f(v_x, v_y) = \begin{cases} 100, & v_x < v_y \\ 010, & v_x > v_y \\ 001, & v_x = v_y \end{cases} . \tag{3.1}$$

Here, the Boolean values on wires w_x and w_y are respectively v_x and v_y, where $v_x, v_y \in \{0, 1\}$ and $f(v_x, v_y) \in \{0, 1\}^3$.

For cases where the function to be computed is publicly known to both parties participating in the protocol, the two parties may jointly construct a circuit that correctly evaluates the function. The structure of this circuit is public knowledge. Then, the parties perform the interactive protocol to learn the result of the calculation while their inputs remain private. This garbled circuit protocol is described in the following section.

3.2.1 The Garbled Circuit Protocol

To illustrate the garbled circuit protocol, this section will consider a scenario with two parties: A and B. The notation used here is loosely derived from the standard notational variations that have

evolved through previous descriptions of the protocol [63, 66–71]. The two parties follow the protocol through four stages. First, both A and B jointly create and agree upon a circuit that evaluates a function of their choice. Second, A generates a "garbled" version of this circuit, essentially encrypting the circuit in such a way that the right set of decryption keys could decrypt the circuit to yield any potential result of the function. Third, the A and B securely exchange the circuit and the necessary inputs; B ultimately receives the decryption keys that solve the circuit correctly for the inputs of both A and B. Fourth and finally, B uses those decryption keys to learn the result of the secure computation. Each of these stages is described in detail below.

Constructing the Circuit

To begin, both A and B agree to jointly compute some function f that can be solved using a circuit $\mathcal{C}$ with m wires and n gates. Let W be the set of all wires $\{w_1, w_2, ...w_m\}$, where w_i is a wire with index $i \in \{1, ...m\}$. Every wire in the circuit carries a value $v_i \in \{0, 1\}$. Similarly, let G be the set of all gates $\{g_1, g_2, ...g_n\}$, where g_j is a gate with index $j \in \{1, ...n\}$.

While many (and likely most) of the circuit wires will both originate and terminate at a gate, the circuit will have some number of wires that define the circuit "edges". Wires that do not originate from a gate will generally be considered input wires of the circuit, cumulatively referred to as the set W_I. Conversely, wires that do not terminate at a gate will generally be considered output wires of the circuit, and will be referred to as the set W_O. Both $W_I, W_O \subset W$, since any useful circuit must have "internal" wires that are neither inputs nor outputs. When the value of every wire in W_I is defined, the values cascade through the circuit, the deterministic Boolean logic gates forcing the values on every wire in W to acquire a defined value. In this case, the result of the calculation is given by the values on the wires in W_O.

Circuit $\mathcal{C}$ represents a jointly evaluated function, and so A and B must each have inputs to the circuit that represent their inputs to function f. Let the input wires that represent the inputs of A be W_A, such that $W_A \subseteq W_I$. Likewise, let the input wires that represent the inputs of B be W_B, such that $W_B \subseteq W_I$. In general, for two parties, $W_I = W_A \cup W_B$.

Since every wire in $\mathcal{C}$ can carry one of two potential values, each gate g_j in circuit $\mathcal{C}$ can be represented by a truth table matching input wire values to the value on the gate's output wire. Boolean logic gates most often accept two input wires (they have a fan-in of two) and always produce one output wire. The relationship between input wire values and output wire values is defined by the operation $g_j \colon \{0, 1\}^2 \mapsto \{0, 1\}$. This produces a table of 2^2 rows, one for each permutation of the input wire values. A set of three example truth tables are depicted in Figure 3.2.

Garbling the Circuit

The garbling process starts with Party A, the garbled circuit generator, assigning two random labels to every wire in circuit $\mathcal{C}$—label $\ell_i^{v_i}$ denotes the label on wire w_i for wire value $v_i \in \{0, 1\}$. Each label is a randomly selected κ-bit binary string $\ell_i^{v_i} \in_R \{0, 1\}^\kappa$. Here, $\in_R$ denotes a uniform random sampling (following the notation of Kolesnikov and Schneider [66]), and κ defines the security-parameter of the circuit. It can be seen in Figure 3.3a that since each row in a truth table possesses

(a) AND

w_x	w_y	w_z
0	0	0
0	1	0
1	0	0
1	1	1

(b) OR

w_x	w_y	w_z
0	0	0
0	1	1
1	0	1
1	1	1

(c) XOR

w_x	w_y	w_z
0	0	0
0	1	1
1	0	1
1	1	0

Figure 3.2: Truth tables defining the behavior of an AND gate, an OR gate, and an XOR gate.

(a) Labeled

w_x	w_y	w_z
ℓ_x^0	ℓ_y^0	ℓ_z^0
ℓ_x^0	ℓ_y^1	ℓ_z^0
ℓ_x^1	ℓ_y^0	ℓ_z^0
ℓ_x^1	ℓ_y^1	ℓ_z^1

(b) Encrypted

Ciphertexts
$E_{\ell_x^0,\ell_y^0}\left(\ell_z^0\right)$
$E_{\ell_x^0,\ell_y^1}\left(\ell_z^0\right)$
$E_{\ell_x^1,\ell_y^0}\left(\ell_z^0\right)$
$E_{\ell_x^1,\ell_y^1}\left(\ell_z^1\right)$

(c) Garbled

Ciphertexts
$E_{\ell_x^1,\ell_y^1}\left(\ell_z^1\right)$
$E_{\ell_x^0,\ell_y^0}\left(\ell_z^0\right)$
$E_{\ell_x^1,\ell_y^0}\left(\ell_z^0\right)$
$E_{\ell_x^0,\ell_y^1}\left(\ell_z^0\right)$

Figure 3.3: Labeled, encrypted, and garbled truth tables for an AND gate.

a unique combination of input wire values, it will also contain a unique set of input wire labels. Since these labels were selected randomly, revealing a label without any additional information confers no information about the associated wire value.

Once wire labels have been used to mask each potential wire value in the circuit, Party A proceeds through each gate, encrypting each row in each truth table. Generally, the encryption process uses some symmetric encryption function $E_{k_1,k_2}\left(\ell\right)$ to encrypt plaintext label ℓ with keys k_1 and k_2. Consider a row in a truth table with input wires w_x and w_y along with output wire w_z carrying value $v_z = g_j\left(v_x, v_y\right)$. The encrypted output wire label is given as a ciphertext token $\mathfrak{e}_{v_x,v_y}$, defined as

$$\mathfrak{e}_{v_x,v_y} = E_{\ell_x^{v_x},\ell_y^{v_y}}\left(\ell_z^{v_z}\right) \ . \tag{3.2}$$

A fully encrypted table is illustrated in Figure 3.3b. While each output wire label is obscured through encryption, the token's position still betrays information about its associated wire value. This association can be removed by shuffling the ciphertexts, removing all apparent correlation between the tokens and the values they represent. A table of permuted ciphertext tokens is depicted in Figure 3.3c. Given exactly one label from each input wire to a gate, an onlooker could only successfully decrypt one ciphertext. That decrypted output wire label would match the truth table row containing those input wire labels. No other ciphertexts could be decrypted, and without more knowledge, the shuffling process prevents that onlooker from learning anything about the label that

was decrypted. This process is repeated for each gate in the circuit, and the full collection of garbled truth tables and tokens constitutes the "garbled circuit" [60]. Note that B has played no role the garbling process.

As mentioned in the introduction to the protocol, the complete garbled circuit may be thought of as an encrypted version of the function represented by the circuit. Given the decryption keys for some circuit input values—specifically the labels on the input wires for those values—the circuit can be decrypted to yield the result of the calculation. The process of securely exchanging those wire labels (the decryption keys) and decrypting the circuit is what allows the garbled circuit protocol to maintain privacy.

Exchanging the Circuit and Inputs

Party A, the garbled circuit generator, may now share the garbled circuit and necessary input wire labels with B, the garbled circuit evaluator. First, A shares the garbled circuit. For each gate in $\mathcal{C}$, A passes B the corresponding garbled table of encrypted tokens. Sharing the garbled tables in this way reveals no information about either party's private inputs; the garbled tables merely contain encodings for the complete set of possible circuit configurations. However, to complete the MPC protocol by decrypting the circuit, B must possess all of the input wire labels corresponding to the input values for both parties. These input wire labels must be transferred in a manner that guarantees the privacy of both parties.

For A, passing the labels for its private input values to B is a straightforward procedure. These wire labels—namely $\{\ell_i^{v_i} \mid w_i \in W_A\}$, where A's choice of input value is v_i on wire w_i—may be transferred directly to B. Since all of the wire labels were randomly chosen and assigned by A alone (unseen by B), any input wire labels that B receives could represent either Boolean value that could be carried on the wire. On the other hand, B must use a clever workaround to receive the labels corresponding to its own input wire values from A; a direct transfer does not preserve privacy. If B were to ask A for the labels corresponding to its choice of input wire values, then A learns this bit choice. Conversely, if A were to transfer *both* potential labels to B, then B could use those two labels together with the single label corresponding to A's input to decrypt tokens for multiple truth table rows.

Instead, A and B initiate an oblivious transfer (OT) protocol to transmit B's labels privately. Using OT, one party (the sender) provides multiple messages as its inputs to the OT protocol, while the second party (the receiver) provides its choice of message to receive as its input. The algorithm returns *only* the chosen message to the receiver, and it returns nothing[3] to the sender [72].

Generally, OT is possible with many input data choices. 1-out-of-n OT, denoted $\binom{n}{1}$OT, corresponds to OT performed with n choices entered into the protocol by the data provider.[4] However, since garbled circuit wires have Boolean values, the simpler 1-out-of-2 OT suffices for exchanging

[3]Equivalently, some OT constructions return the null string (which is arguably nothing) back to the sender providing the messages.

[4]The binomial coefficient notation distinguishes this "chosen 1-out-of-n OT" construction from the alternative presented by Even, Goldreich, and Lempel where the receiver randomly receives one of the n inputs, each with equal probability [72].

wire labels between A and B. Using $\binom{2}{1}$OT for an input wire $w_i \in W_B$ carrying one of B's inputs, A provides both potential wire labels (ℓ_i^0, ℓ_i^1) to the OT algorithm, and B provides the value v_i, its choice of which label to receive. The two parties proceed to engage an interactive protocol that results in an oblivious transfer of data; B receives the one wire label corresponding to its input value on wire w_i, and A learns nothing about which label B received.

The exact mechanism behind the OT construction is not generally important in the operation of garbled circuits, and OT is often treated as a black box algorithm. Various methods exist to accomplish OT [72–76], though a popular and reasonably intuitive method for performing $\binom{2}{1}$OT was suggested by Even, Goldreich, and Lempel [72]. In that version of $\binom{2}{1}$OT, public-private key cryptography is used to mask the input choice of the circuit evaluator. Party A chooses two random label "masks", one for each of its inputs to the OT protocol, and also generates a public-private key pair. Both masks and the the public key are shared with B. Party B then generates a mask of its own, which it encrypts using the public key. At this point, B makes its selection of which label to learn; it uses the mask provided by A matching that selection to encrypt the already-encrypted token it just created using the public key. Sending this twice-encrypted token back to A, A uses all of the information in its possession to attempt to decrypt the token. First, A applies each of the masks it generated to produce two potential versions of the evaluator's mask encrypted under the public key. Then, it uses the private key to decrypt both of these versions, giving it the mask generated by B, as well as a bogus mask. Since the mask B generated was random, A should not be able to distinguish the true mask from the bogus mask. At this point, A can use the two masks to encrypt the two input labels. Party A sends these masked labels to B, who can then use its own mask to decrypt the label of its choice. Since B does not know the private key, it has no way to learn the bogus mask, and so it is unable to decrypt anything more than the label it chose.

Finally, after completing the OT protocol, B is in possession of two labels for each circuit input wire: one corresponding to A's unknown input value, and one corresponding to its own known input value.

Evaluating the Circuit

Equipped with labels representing the true inputs of both parties for the set of input wires W_I, party B can now decrypt exactly one output wire label for each gate in $\mathcal{C}$. To perform this decryption, the evaluator must know the symmetric decryption function D_{k_1,k_2} corresponding to the symmetric encryption function E_{k_1,k_2} used by the circuit generator. This decryption function must satisfy the relation

$$D_{k_1,k_2}\left(E_{k_1,k_2}\left(\ell\right)\right) = \ell\,. \tag{3.3}$$

Party B will begin by using D and the set of input wire labels to decrypt output wires for all gates connected to the circuit's input wires. Then, the decrypted labels on the output wires from those gates are harnessed to decrypt subsequent gates in the circuit. This process continues until all gates in the circuit have been decrypted.

The literature contains suggestions for multiple methods of identifying a successful decryption. Most simply, a guess-and-check technique can be applied to the encrypted tokens in any order. If

some indicator is appended to each label prior to encryption—for example a string of σ consecutive zeros—then a successful decryption can be identified with high probability[5] as the only decryption result with that suffix. If the evaluator randomly decrypts labels until encountering the designated output label, they will perform an average of 2.5 decryptions per gate, the expected value of sampling one, two, three, or four decryption attempts with equal probability. To improve efficiency, it is common to use the point-and-permute method, described with more detail in Section 3.2.2.

Finally, once the circuit evaluator B learns the true labels associated with all of the circuit output wires W_O, the two parties may come together to learn the calculation result. If A exposes the mapping between output wire labels and values and B reveals the decoded output wire labels, the two may both learn the result of the calculation [63]. In this simple version of the garbled circuit protocol, these final exposures do not forfeit any privacy, since by definition MPC allows both parties to know the result of the joint calculation, but nothing more.

3.2.2 Optimizations for Yao's Garbled Circuits

In their most basic form, garbled circuit protocols are computationally expensive. Examining the circuit constructions described earlier, it can be deduced that a garbled circuit essentially creates a structure that is able to represent every possible configuration of a system—with only one of these possibilities being jointly solved by the computation's participants. With that in mind, significant effort has been directed towards optimizing garbled circuit protocols to reduce their computational burden. These modifications tend to save time or storage requirements, either by reducing the number of gates that must be encrypted, reducing the number of encryption operations required by a given type of gate, or reducing the quantity of information that must be communicated between parties.

Point-and-Permute

One commonly used optimization eliminates the need for the guess-and-check decryption procedure described in Section 3.2.1. This procedure, called *point-and-permute*,[6] assigns a pointer to each wire label such that when input wire labels to a gate are known, they directly indicate which output label should be decrypted [60].

For Boolean circuits, this pointer need only be one bit and is commonly referred to as the permutation bit [66, 70, 77].[7] If $p_i^0, p_i^1 \in \{0, 1\}$ represent the permutation bits for wire w_i, then the full keys used to encrypt the output wire label (and permutation bit) are $k_i^{v_i} = \langle \ell_i^{v_i} \parallel p_i^{v_i} \rangle$. Here, the operator $\parallel$ denotes string concatenation (such that if a label ℓ is a κ-bit binary string and a permutation

[5] If D returns decryptions that are uniformly distributed, then the likelihood that any incorrect decryption also ends in σ zeros is $2^{-\sigma}$.

[6] Evans, Kolesnikov, and Rosulek note that this name became popular around 2010, well after the technique's introduction [63].

[7] It is also occasionally named the select bit [78], signal bit [70], or color bit [79, 80].

bit p is a single bit, then the associated key k is a $(\kappa + 1)$-bit string).[8] To avoid disclosure of any information about a label's associated value, the permutation bits must be chosen independently of the labels. This independence can be achieved by randomly selecting one permutation bit, say $p_i^0 \in_R \{0, 1\}$. Then, since both permutation bits must be different, $p_i^1 = \neg p_i^0$, where $\neg$ denotes a logical negation.

Given this construction, each row in a gate's truth table will have a unique set of both labels and permutation bits. Because each pair of permutation bits was selected randomly, ordering the garbled table of ciphertext tokens by the ordered tuple of input wire permutation bits in each row constitutes a random permutation of the table. Once the circuit evaluator possesses the keys necessary to decrypt a gate's truth table, they use the two labels to decrypt the token at the index defined by the permutation bits. As an example, consider the case of a circuit evaluator possessing keys $k_a^{v_a} = \langle \ell_a^{v_a} \parallel p_a^{v_a} \rangle$ and $k_b^{v_b} = \langle \ell_b^{v_b} \parallel p_b^{v_b} \rangle$ to a standard Boolean gate with fan-in two—a gate with input wires w_a and w_b. The evaluator also has access to a garbled table of tokens for that gate, where token e_{v_a,v_b} has been placed in the sorted table according to its ordered pair of permutation bits $(p_a^{v_a}, p_b^{v_b})$. The evaluator's keys point them directly to this entry in the garbled table, and so no more than one decryption is required for any gate. That is less than half the average number of decryptions required for the guess-and-check method described in Section 3.2.1

Row Reduction

The number of encryptions required per gate during the garbling process can also be reduced using a technique named *row reduction*, a concept introduced in by Naor, Pinkas, and Sumner [81]. In traditional garbling schemes, all wire labels are selected from a uniform random distribution of κ-bit strings $\{0, 1\}^\kappa$, and are thus independent of the encryption function E. Instead, to administer the garbled row reduction optimization, one output wire label for each gate g_i may be chosen such that the result of one encryption is a known constant. For example, let the ciphertext token corresponding to the truth table row with input wire values v_a and v_b be chosen for reduction. An encryption function E would be chosen such that the ciphertext would be a string of zeros with some arbitrary length:

$$\mathfrak{e}_{v_a,v_b} = E_{\ell_a^{v_a},\ell_b^{v_b}}\left(\ell_c^{g_j(v_a,v_b)}\right) = \{0\}^* . \tag{3.4}$$

While identifying encryption inputs satisfying this relationship may be challenging for some encryption functions, it is straightforward for any E based on the one-time pad.[9]

[8] Alternatively, Ball, Malkin, and Rosulek suggest that rather than affixing the permutation bit to the κ-bit label, the label might be truncated, with the permutation bit(s) replacing the truncated digits [79]. This variation preserves the length of the key as κ, which is often advantageous when executing garbled circuits in practice that use cryptographic primitives requiring κ-bit inputs (e.g. 128-bit AES or the SHA-256 hash function). The authors note that this minor degradation process is used in all implementations of which they were aware.

[9] The one-time pad encryption scheme simply uses a κ-bit key k to encrypt a message m of the same length by applying the exclusive or (XOR) operation to each bit: $\mathfrak{e} = m \oplus k$.

w_a	w_b	w_c
ℓ_a^0	ℓ_b^0	$\ell_a^0 \oplus \ell_b^0$
ℓ_a^0	$\ell_b^0 \oplus \Delta$	$\ell_a^0 \oplus \ell_b^0 \oplus \Delta$
$\ell_a^0 \oplus \Delta$	ℓ_b^0	$\ell_a^0 \oplus \ell_b^0 \oplus \Delta$
$\ell_a^0 \oplus \Delta$	$\ell_b^0 \oplus \Delta$	$\ell_a^0 \oplus \ell_b^0$

Table 3.1: The labels dictated by the *FreeXOR* technique for an XOR gate.

FreeXOR

One of the most consequential garbled circuit optimizations is the *FreeXOR* technique presented by Kolesnikov and Schneider [66]. The premise of this optimization strategy is recognizing that for common gates with even output parity[10]—XOR and XNOR gates, each with two values of zero and one in the output column of their respective truth tables—the relationship between truth table labels and values is naturally obscured [63]. Intuitively, this is the concept exploited by the information-theoretic one-time pad, where any given output wire has the potential to be associated with either label on a given input wire.

Using the *FreeXOR* technique, XOR and XNOR gates can be securely encoded and evaluated with simple computational instructions, entirely avoiding the significantly more expensive burden imposed by encryption. To implement the technique, a global key offset Δ is first assigned to a circuit by the circuit generator. The global offset is a randomly selected string in the length of the security-parameter: $\Delta \in_R \{0,1\}^\kappa$. Next, labels are randomly selected for all values of zero on input wires to the circuit, similar to the standard garbled circuit protocol:

$$\left\{\ell_i^0 \in_R \{0,1\}^\kappa \mid w_i \in W_I\right\} \ . \tag{3.5}$$

Labels paired with the values of one on the circuit's input wire are not selected at random, however, and are instead derived from the zero-value wire labels using the global key offset:

$$\left\{\ell_i^1 = \ell_i^0 \oplus \Delta \mid w_i \in W_I\right\} \ . \tag{3.6}$$

After determining labels for all of the circuit's input wires, the circuit generator proceeds through the circuit gate-by-gate. For gate g_j in G, if g_j is either an XOR or XNOR gate, the labels on the output wire are calculated directly from the input wire values. If gate g_j has input wires w_a and w_b along with output wire w_c, then for any set of values v_a, v_b, and v_c, the corresponding output wire label is given by $\ell_c^{v_c} = \ell_a^{v_a} \oplus \ell_b^{v_b}$. The full arrangement of values is shown in Table 3.1. Just as the circuit generator directly calculates all of the output wire labels using the input wire labels, so too may the circuit evaluator calculate the output wire label corresponding to whichever input wire

[10]"Common" gates is a key qualifier; a gate that follows or reverses just one of the input wires could not be used with the *FreeXOR* technique despite having the "even" quality described by Evans, Kolesnikov and Rosulek, but these gates are highly uncommon and generally unnecessary in MPC.

labels they are given. Since the XOR process is not computationally intensive compared to other expensive cryptographic primitives, the computation required to prepare or evaluate any of these gates is essentially free when compared to non XOR/XNOR gates.

For gates that are not "free", the circuit generator follows a different procedure. Since the input wire labels for these gates cannot be used to produce the output wire labels securely through the same XOR operation, the output wire labels must be freshly generated. This follows the same procedure used by the circuit's input wires, where the gate's output wire label for value zero is randomly selected while the label for value one is derived using the global key offset (reproducing Equations 3.5 and 3.6, where w_i is now the output wire of gate g_j).

From there, the garbling process remains fairly similar to the standard garbled circuit protocol, though care must be taken to ensure the protocol's security. Since the circuit's labels are no longer independent—all related to one another via the global key offset—it is imperative that the global key cannot be reconstructed by the circuit evaluator.

First, this means that the encryption process must use input labels as symmetric encryption keys in such a way that no correlation exists between the keys and the resulting ciphertext. Encryption of this sort may be accomplished by using a random oracle (RO) to produce a random, reproducible string for any given set of concatenated input wire labels. The resulting ciphertext can be used as a one-time pad to encrypt the output label. Critically, since the same pair of wires could conceivably be used as inputs to different gates, the RO should also accept the index j of the gate being garbled along with the concatenated input labels [82]. Otherwise, the RO would return the same value for two different gates constructed using the same set of input wires, rendering the one-time pad insecure. Explicitly, the equation

$$\mathfrak{e}_{v_a,v_b} = H\left(\ell_a^{v_a} \parallel \ell_b^{v_b} \parallel j\right) \oplus \ell_c^{g_j(v_a,v_b)} \tag{3.7}$$

defines the ciphertext token for the row with input wire values v_a and v_b [66]. Function $H\colon \{0,1\}^* \mapsto \{0,1\}^\kappa$ represents the RO, usually implemented in practice by a hash function (e.g. SHA-1 or SHA-256) [66, 82].

Second, since the circuit evaluator must be prevented from learning the global key offset, the final exposure step of the traditional garbled circuit protocol must be modified. If this were not the case, the circuit evaluator could reconstruct the global key as soon as the circuit generator revealed the mapping between output wire labels; they would know both labels on at least one wire, and those labels would differ only by the offset. Instead, the circuit generator can conceal the output wire label-value relationship using a method analogous to the one used for encrypting garbled tables. For each output wire $w_i \in W_O$, the generator creates the ciphertext token

$$\mathfrak{e}_{v_i} = H\left(\ell_i^{v_i} \parallel \text{`out'} \parallel i\right) \oplus v_i\,. \tag{3.8}$$

This procedure masks each output value using its label, such that a circuit evaluator may reveal the output wire value. Without ever learning the other output wire label or enough to deduce the global key offset, the *FreeXOR* method is secure.

Finally, the decryption process for a circuit that has been encoded using the *FreeXOR* technique is straightforward. Once the circuit evaluator possesses labels on each input wire, either through

direct exchange or via OT, the evaluator proceeds gate by gate as in the traditional protocol. For gates that are "free", the evaluator simply computes the XOR of the gates' input wires. For every other gate (and the output wires), the evaluator can evaluate H for the input wire labels it possesses and the public gate index, exploiting the property that the one-time pad is a self-inverse encryption scheme. The evaluator repeats this process until the have decrypted all gates and have learned the circuit outputs.

The *FreeXOR* technique also benefits from the highly valuable property that it is compatible with other optimization techniques; it can be used with both the point-and-permute and row reduction optimizations. First, point-and-permute can be made to work with *FreeXOR* by noting that the two permutation bits on any wire are opposites, $p_i^1 = \neg p_i^0$, and that this may be equivalently expressed by taking the XOR of either bit and one: $p_i^1 = p_i^0 \oplus 1$. Expressing the relationship between permutation bits in this way allows Equation 3.6 to be extended such that this constant is appended to the global key offset:

$$\left\{ k_i^1 = k_i^0 \oplus \langle \Delta \parallel 1 \rangle \mid w_i \in W_I \right\} \tag{3.9}$$

where $k_i^{v_i} = \langle \ell_i^{v_i} \parallel p_i^{v_i} \rangle$. This key value $k_i^{v_i}$ is used in place of label $\ell_i^{v_i}$ in the remainder of the *FreeXOR* instructions. To combine *FreeXOR* with the row reduction technique, the output labels from non-free gates must be chosen such that both Equations 3.4 and 3.8 are both satisfied. Specifically, this is

$$\ell_c^{g_j(v_a,v_b)} = H\left(\ell_a^{v_a} \parallel \ell_b^{v_b} \parallel j\right) , \tag{3.10}$$

since the XOR of any bit and itself is zero.

3.3 Other Circuit Constructions

Yao's garbled circuit became an attractive MPC protocol due to the efficiency of evaluation and communication. As described earlier, garbled circuits only require OT operations to exchange inputs for one of the parties, and otherwise can be constructed and evaluated using fast symmetric encryption. Other circuit-based MPC protocols exist, however, each with their own merits that best fit different situations. Some alternative protocols are presented here for completeness, but they will not be used for any of the work presented in later chapters. It is still important to consider these variations when discussing MPC applications in nuclear safeguards in general though, as there are likely analysis contexts where they would find substantial utility.

3.3.1 GMW Circuits

A popular alternative to Yao's garbled circuit protocol is the Goldreich-Micali-Wigderson (GMW) protocol. Like garbled circuits, the GMW protocol is formulated using digital logic circuits [55, 63]. However, the two methods differ in their methods for sharing and evaluating gate values.

Where the garbled circuit protocol assigns κ-bit labels to each wire to use as symmetric encryption keys, the GMW protocol uses a secret sharing scheme to compute single bit shares to obfuscate the wire values. Specifically, in the case of two parties A and B, each party generates a random bit

$S\left(s_a^B, s_b^B\right) = v_c$
$S(0,0) = \left(s_a^A \oplus 0\right) \wedge \left(s_b^A \oplus 0\right)$
$S(0,1) = \left(s_a^A \oplus 0\right) \wedge \left(s_b^A \oplus 1\right)$
$S(1,0) = \left(s_a^A \oplus 1\right) \wedge \left(s_b^A \oplus 0\right)$
$S(1,1) = \left(s_a^A \oplus 1\right) \wedge \left(s_b^A \oplus 1\right)$

Table 3.2: The four potential configurations that produce v_c from the perspective of A. Shares s_a^A and s_b^A are considered fixed since they are known to A.

mask $r_i \in_R \{0,1\}$ for each input wire w_i where they know their own choice of value (the sets W_A and W_B, respectively). From these bit masks, a party may compute two shares for the value on the i^{th} wire: $s_i^A = r_i$ and $s_i^B = r_i \oplus v_i$ such that $v_i = r_i \oplus (r_i \oplus v_i) = s_i^A \oplus s_i^B$. The parties exchange shares while keeping one share for itself; A gives each share s_i^B to B and B gives each share s_i^A to A. No secure information is revealed in this procedure, since all shares are derived from random number r_i and so are themselves randomly distributed.

The circuit is then evaluated gate by gate. For each gate in the circuit, the value on the gate's output wire may be calculated by anyone knowing all of the shares to the gate's input wire values. As an example, consider an `AND` gate where the output value is $v_c = v_a \wedge v_b = \left(s_a^A \oplus s_a^B\right) \wedge \left(s_b^A \oplus s_b^B\right)$. Knowing all four share values s_a^A, s_a^B, s_b^A, and s_b^B enables direct calculation of v_c, which could then be split into shares s_c^A and s_c^B. When the gate's output wire is dependent on the private inputs of both parties, neither party can ever know all of the shares. Otherwise, they would be able to reconstruct the private values of the other party and there would be no security.

Instead, the parties need to find a way to exchange the shares s_c^A and s_c^B without revealing their private information. OT provides the solution. The exchange process begins with one party computing the complete set of potential values for v_c using their own known shares and the possible shares of the other party. Let A be the first party. From this A's perspective, there are no more than four configurations that could produce v_c. This is due to the fact that there are up to two shares being held by B, which are unknown to A, along with the fact that each share could be either a zero or a one. For A, the potential value v_c can now be expressed as a function dependent on B's shares: $v_c = S\left(s_a^B, s_b^B\right)$. Table 3.2 illustrates these configurations from the perspective of A, where s_a^A and s_b^A are known and thus essentially fixed. Like with the input wires, each potential value of v_c may then be converted into shares by A—namely $s_c^A = r_c$ and $s_c^B = r_c \oplus S\left(s_a^B, s_b^B\right)$ for each permutation of s_a^B and s_b^B. Then, A and B engage in 1-out-of-4 OT, where A provides the four possible values of s_c^B as choices and B provides the true values of s_a^B and s_b^B as its choice. B ultimately learns share s_c^B for wire w_c and A learns nothing about B's choice.

While this process could work for any arbitrary gate, two classes of gates can be solved in a more efficient manner. The associativity of the XOR operation enables the `XOR` gate to be evaluated directly, without requiring OT. Now, where

$$v_c = \left(s_a^A \oplus s_a^B\right) \oplus \left(s_b^A \oplus s_b^B\right) = \left(s_a^A \oplus s_b^A\right) \oplus \left(s_a^B \oplus s_b^B\right) \ , \tag{3.11}$$

A and B can compute the first and second terms, respectively, as their shares of v_c; no OT is necessary. Likewise, `NOT` gates can also be simplified. Inverting the secret-shared value can be accomplished trivially by inverting one of the two shares. For example, a `NOT` gate with input wire value v_i composed of shares s_i^A, s_i^B (one each held by parties A and B) has output value $v_i \oplus 1$.[11] This value would be properly represented by shares $s_i^A \oplus 1$ and s_i^B, where $s_i^A \oplus 1$ can be computed exclusively by A. Again, no OT is required. All other logic gates may be implemented either using these techniques or by recognizing that the `AND`, `XOR`, and `NOT` gates represent a set of universal logic gates [83: §10.1.6A]. These three methods of gate evaluation are sufficient for solving any GMW Boolean circuit.

Although this scheme affords a reduction in complexity, the tradeoff is that the GMW protocol requires a 1-out-of-4 OT per gate, as opposed to the 1-out-of-2 OT per evaluator input wire required in Yao's garbled circuit protocol. On the other hand, observing that expensive OT operations may be largely precomputed during an offline phase before commencement of the interactive protocol may make GMW a more attractive option than Yao's garbled circuit protocol in certain circumstances [84].

3.3.2 Arithmetic Circuits

Arithmetic circuits present another alternative to the traditional garbled circuit protocol. Traditional garbled circuits, as well as GMW circuits, are constructed based on the notion that every wire in the circuit may possess one of two possible values—a zero or a one. While this concept is intuitive and mimics the natural world where digital circuits have wires with Boolean values, circuits represented by virtual circuits are not subject to this limitation. Instead, the wires in arithmetic circuits may be multivalued, a property that allows single gates in the circuit to perform more elaborate mathematical operations than Boolean gates.

A simplified notion of arithmetic circuits can be intuited by extrapolating from Boolean circuits. In a Boolean circuit, each wire can take one value in base two; wire w_i was previously defined to carry value $v_i \in \{0, 1\}$, which could be equivalently written $v_i \in \mathbb{Z}_2$. Here $\mathbb{Z}_2$ represents the congruence class of integers modulo 2. When considering Boolean circuits in this way, an `XOR` gate can be interpreted as performing addition (modulo 2) while an `AND` gate can be interpreted as performing multiplication (modulo 2).[12]

For arithmetic circuits, this concept can be generalized to integer rings modulo m. Where wires in the Boolean circuit were restricted to carrying values in $\mathbb{Z}_m$, wires in an arithmetic circuit may carry any value in $\mathbb{Z}_m$ [79]. Where labels in a Boolean circuit with security parameter κ are $\ell_i^0, \ell_i^1 \in \{\mathbb{Z}_2\}^\kappa$ (now adopting the algebraic ring notation for the set of base two integers), labels in an arithmetic circuit are assigned as $\ell_i^0, \ell_i^1 \in \{\mathbb{Z}_m\}^{\kappa_m}$. To maintain the same level of security (same label length) as a Boolean garbled gate with security parameter κ, an arithmetic circuit must have $\kappa_m = \lceil \kappa / \log_2(m) \rceil$ [79].

[11] Inversion can be considered a special case of the XOR operation when one of the inputs has a fixed value of one.
[12] Figures 3.2a and 3.2c are illustrative.

Additionally, this generalization suggests that some of the optimizations developed for Boolean circuits might also be extended to arithmetic circuits. This turns out to be the case, and where Boolean circuits can take advantage of the *FreeXOR* technique to reduce the number of ciphertexts that must be exchanged between parties, so too can arithmetic circuits take advantage of the equivalent operation in $\mathbb{Z}_m$ to perform addition (modulo m) for "free". Given a wire w_i with label $\ell_i^0 \in_R {\mathbb{Z}_m}^{\kappa_m}$ along with a global key offset $\Delta_m \in_R {\mathbb{Z}_m}^{\kappa_m}$ (both the label and offset are vectors of $\mathbb{Z}_m$-elements), the complete set of labels on wire w_i can be given as

$$\left\{ \ell_i^{v_i} = \ell_i^0 + v_i \Delta_m \mid v_i \in \mathbb{Z}_m \right\} . \tag{3.12}$$

Then, given a gate evaluating the sum v_c of any two input wire values $v_a, v_b \in \mathbb{Z}_m$, a party knowing two input wire labels $\ell_a^{v_a}$ and $\ell_b^{v_a}$—but who does not know v_a and v_b—may still be able to calculate the corresponding output label:

$$\ell_c^{v_c} = \ell_c^0 + v_c \Delta_m = \left(\ell_a^0 + \ell_b^0 \right) + \left(v_a + v_b \right) \Delta_m = \ell_a^{v_a} + \ell_b^{v_b} . \tag{3.13}$$

Using the same form, arithmetic circuits may also perform "free" multiplication by a publicly known constant, as well as implement the point-and-permute optimization technique [79].

Arithmetic circuits offer a significant efficiency enhancement in circuits that rely primarily on mathematical calculations, such as addition or multiplication, because addition operations can be performed with minimal computational cost [79, 85]. This behavior is especially promising when considering that performing calculations in traditional formats requires inputs that are several bits in length (e.g. 32-bit or 64-bit integers, floating point numbers, and strings). Constructing the Boolean circuit equivalents of these circuit components frequently requires complex gate structures and therefore many encryption iterations during the garbled circuit protocol. Arithmetic circuits enable many of these operations essentially for "free". Despite this advantage, arithmetic circuits require substantially more encryption iterations to evaluate Boolean constructs (e.g. comparisons or equality tests) than the equivalent Boolean circuits. In this way, MPC protocol designers must consider the types of calculations to be performed when selecting a circuit style to use. To leverage the advantages of each circuit type, recent work has focused on developing hybrid style circuits that use arithmetic circuit evaluation to perform mathematical calculations while using the traditional Boolean circuits for executing operations like comparisons more efficiently [86].

3.4 Malicious Adversaries

Up until this point, discussion has focused on passively secure protocols. As described in Section 3.1, these protocols are secure when the involved parties can be trusted to execute the protocol faithfully. These semi-honest parties may inspect every piece of information that they are given in an attempt to learn as much as possible about the calculation, but they will not subvert the protocol to either produce the wrong answer or violate another party's privacy.

This adversarial model may be appropriate in some situations, for example when parties may not have motivation to cheat or when modifying the protocol is unlikely or cost-prohibitive. A MPC

protocol that is used to share sensitive personal information with a generally trustworthy third-party agency may fall into this classification. Consider the case of a research study, similar to that performed by a team at Boston University in 2016 [87]. A research institution wants to use the data held by many private companies to draw broad conclusions about the state of society. Each individual institution has an interest to protect their data, whether to retain proprietary advantages, uphold ethical or legal commitments to its clients, or preserve its reputation. The individual companies see no reason for the research institution to act maliciously and learn more than the aggregate information needed for the study. Likewise, the research institution has no reason to believe that the companies—who have been guaranteed privacy—should attempt to manipulate the final result. The parties may then consider all other parties to be semi-honest, and a passively secure model is adequate. In other cases, an MPC protocol that is sufficiently difficult to modify may be justifiably passively secure. Examples here include protocols implemented by a closed-source application, or a protocol implemented by parties who do not expect the other parties to possess the time, resources, or expertise to introduce vulnerabilities.

For nuclear safeguards scenarios, the passively secure model may be sufficient in a limited set of circumstances, but otherwise a stronger model should be considered. Although the safeguards inspectorate may be in a position similar to a trusted agency, it is far less likely that a facility under safeguards would be completely trusted by the regulator. If this were the case, safeguards would be unnecessary in the first place.

Instead, there are MPC protocols designed to provide stronger assurances of security against more challenging adversarial models. For the strongest assurances against unscrupulous behavior, parties may engage in *actively secure protocols*. These include techniques like the cut-and-choose technique and a batched variant, along with a technique based on ZKPs called the GMW-compiler.

The cut-and-choose technique is one of the earliest and most popular strategies for guaranteeing security against malicious adversaries. Starting from the traditional garbled circuit protocol, the cut-and-choose method requires the circuit generator to construct several garbled circuits and garble each independently. These circuits are all passed to the circuit evaluator, and a random subset of circuits is selected to be checked. The circuit generator reveals the potential labels on all of the circuit wires (as well as the global key offset, if used), so that the evaluator can see that each of the circuits was indeed garbled correctly. In the event that the evaluator finds that a circuit was generated incorrectly, they become aware that the generator attempted to cheat and can abort the protocol before continuing.

To ensure that the probability that an improperly garbled circuit goes undetected is negligible, all of the remaining circuits left unchecked may be evaluated on the two parties' real inputs. These evaluated circuits should all produce the same answer, and any inconsistencies indicate malicious behavior. In spite of this, the evaluating party should not abort the protocol at this point, even if inconsistencies are detected, as terminating the protocol could reveal information about their own inputs.[13] Instead, the evaluator should determine the output given by the majority of the evaluated circuits and consider that to be the true output. The probability of generating a subset of improperly

[13]The generator could devise a malicious implementation of a circuit that only produces inconsistent outputs for certain evaluator inputs. Then, the evaluator would disclose information about its input simply by aborting the protocol.

garbled circuits—none of which are checked but all of which represent a majority of the remaining evaluated circuits—is vanishingly small. When evaluating the circuit, care must be taken to ensure that the circuits are evaluated using consistent inputs by both parties lest either party attempt to circumvent the defenses provided by the cut-and-choose technique. Since the cut-and-choose process introduces substantial computational overhead to a MPC calculations, other variations of cut-and-choose techniques have been introduced to enhance efficiency. For repetitive garbled circuit calculations, a batched version of the cut-and-choose method has been proposed, allowing parties to evaluate several batches of garbled circuits (rather than evaluating all circuits with the same inputs). Security guarantees remain strong in this case, but only one round of circuit checks must be performed for all of the evaluated batches.

An alternative to the cut-and-choose technique for providing active security is to rely on a compiler construction first introduced by Goldreich, Micali, and Wigderson in 1987. By using a ZKP to prove honest execution of each step in a MPC protocol, the GMW compiler enables any semi-honest protocol to be converted into an protocol secure against malicious adversaries. Through commitments to their initial input values and expected randomness, each party proves to the other at every step that the protocol so far has proceeded according to the protocol's rules.

In many cases, these actively secure protocols may be even stronger than what is strictly necessary, and so slightly weaker *covertly secure protocols* may be sufficient. While actively secure protocols detect cheating in all but a negligible fraction of attempts—effectively preventing a cheating party from *ever* being successful—covertly secure protocols instead reveal malicious activity with some significant probability. In many cases, just the possibility of exposure would serve as a deterrent against poor behavior [65].

On the whole, these covert protocols have a great deal in common with actively secure protocols. For instance, the covert technique outlined by Aumann and Lindell [65] is quite similar to the cut-and-choose method. This technique still sees the circuit generator providing a selection of garbled circuits to the evaluator, but now, rather than selecting subsets for checking and evaluating so that the chance of undetected malicious activity is negligible, the evaluator would simply challenge the generator to reveal the randomness used to generate all circuits except those selected for evaluation. Aumann and Lindell define the likelihood that malicious activity is detected as the deterrence factor ϵ [65]. If a generator creates N garbled circuits with only one circuit garbled maliciously, then there is a $\epsilon = 1 - \frac{1}{N}$ chance that the dishonest circuit will be exposed. Again, similar to the case of actively secure models, care must be taken to protect the protocol from subtle weaknesses, for example by using maliciously or covertly secure OT protocols, ensuring consistent inputs, and executing the protocols in an order that disallows exploitation in future steps [63].

A covertly secure protocol would likely be ideal for a safeguards approach. While less computationally intensive (and therefore, less expensive) than an actively secure protocol, a covertly secure protocol that exposed impropriety on the part of a safeguarded facility could trigger more extensive investigations and consequences from the IAEA. These penalties could follow the model of those that would be meted out for any other safeguards violation. Furthermore, the probability threshold for detecting malicious activity used by a covertly secure protocol could be tailored to match the IAEA's technical objectives, as discussed in Section 2.3.1. At the same time, unless computational resources are abundant, actively secure protocols may be prohibitively expensive for systems that

are designed to augment an existing safeguards setup.

3.5 MPC Frameworks

As the theory of MPC has matured, a wide variety of MPC specific frameworks have emerged. In general, each framework tends to have a specific purpose, typically to showcase a recent efficiency improvement or unify existing methods. Some notable frameworks and MPC tools include Fairplay/Fairply-MP, TinyGarble and its successor JustGarble, ABY, Viff, Frigate, SPDZ/MP-SPDZ, Sharemind, and Obliv-C, although many others exist; a selection of these frameworks are described and referenced below.

Among the first documented public MPC codes was the Fairplay computation tool [82]. The Fairplay engine provides users with the ability to perform MPC using Yao's garbled circuit protocol, providing an application framework to facilitate the interaction. Realizing that Boolean circuit assembly was a significant burden for most MPC users, Fairplay introduced a secure function definition language (SFDL) in which secure applications could be written. To preserve familiarity, the Fairplay SFDL resembled existing software languages. Then, the Fairplay engine would compile a program described in the SFDL into a circuit representation in the Fairplay secure hardware definition language (SHDL)—a compiled circuit representation similar to those produced by VHDL or Verilog hardware description languages. Unfortunately, the memory structure of the Fairplay framework was ultimately found to render the software incapable of successfully evaluating even moderately sized functions [88]. Development on the Fairplay engine continued, however, eventually expanding into FairplayMP. While the original Fairplay system was limited exclusively to two parties, the FairplayMP system could perform computations between several parties [89]. FairplayMP used the Beaver-Micali-Rogaway (BMR) protocol to extend Yao's garbled circuit protocol to implement passively secure calculations among an arbitrary number of parties [60].

Beyond Fairplay, other early MPC software projects sought to implement substantially more powerful protocols or protocols that demonstrated active security against malicious adversaries. One such project was Virtual Ideal Functionality Framework (VIFF), which constructed a framework based on Shamir secret sharing [54] and protecting against malicious adversaries [90]. VIFF leveraged asynchronous communication to minimize communication bottlenecks during interactive protocols, parallelize computation, and minimize protocol complexity (with fewer waiting periods). Rather than the traditional Boolean circuits used by Fairplay, VIFF was constructed using multi-valued circuits, specifically circuits defined for finite fields $\mathbb{F}_m$ [90, 91]. Due to its construction, however, VIFF required at least three participating parties in every calculation. The project was ultimately abandoned in favor of the SPDZ family of MPC software—SPDZ, SPDZ-2, SCALE-MAMBA, and MP-SPDZ [92–95].

SPDZ and VIFF share several key features: both protocols perform computations on finite field circuits, and both protocols have distinct "online" and "offline" execution stages [90, 92]. Still, SPDZ evolved to include stronger security guarantees against more malicious parties (security against a dishonest majority of parties, as opposed to just some threshold number of corrupted parties) [92, 96]. Later iterations of SPDZ continued to improve upon the original SPDZ proto-

col, broadening functionality, improving key generation procedures, and providing stronger, more efficient covert and active security models [93]. Two notable implementations of the SPDZ protocol, both still actively maintained, are SCALE-MAMBA and MP-SPDZ. While SCALE-MAMBA is a security-centric implementation and only provides malicious security, MP-SPDZ (short for "Multi-Protocol" SPDZ) is designed for comparing various MPC protocols—both garbled circuits and secret sharing [94, 95].

Other notable MPC software tools include: JustGarble [97], a garbling tool taking advantage of a fixed-key blockcipher; ABY [86], an MPC framework that blends arithmetic, Boolean (secret-shared), and Yao's garbled circuits; TinyGarble [98], a software system to garble circuits in a compact and optimized fashion; and Obliv-C [99], a language extended from C intended to facilitate the use of garbled circuits by general programmers.

Despite the sheer number of MPC techniques and software frameworks that have been published and distributed, there is no clear ideal framework [69]. This is unsurprising given the distinct capabilities of each tool; however it is a software landscape that may prove challenging for those without experience in the field who attempt to select an MPC framework. Making things even more difficult, is the fact that many of the published codes cited in the literature have been replaced or ceased to be maintained, and even active MPC software often fails to be documented in a manner that encourages third-party usage.

Along these lines, while several attempts have been made to benchmark and compare the codes in terms of performance, far less attention has been devoted to understanding how communities of non-expert users might interact with the software. In one such attempt to characterize the usability of a broad collection of recently developed MPC tools, even a team of computer scientists struggled in several cases to reproduce even simple calculations [100]. That study determined that most of the MPC codes available exist in different lifecycle phases and with highly varied levels of usability.

It is with this background that this work has chosen the Obliv-C software framework for performing a demonstration of how MPC could be applied to nuclear safeguards. Obliv-C represents one of the most accessible software frameworks available, an open source tool that is designed for developers with only a modicum of experience involving privacy-preserving computation and who are familiar with C and C-like programming languages. The next chapter describes how anomaly detection algorithms similar to those that might be employed by safeguards administrators could be constructed as MPC calculations driven by a privacy-preserving engine like Obliv-C. Then, since even Obliv-C tends to be rather opaque in its runtime execution, Chapter 5 describes the *CypherCircuit* software package, a computational tool that was developed during this work to showcase MPC calculations taking place at a low level. Ideally, the *CypherCircuit* package serves as a simple, intuitive, and useful prototyping tool to convince skeptical parties of the utility and security of MPC technologies.

Chapter 4

Safeguards Anomaly Detection

Where Chapters 2 and 3 proposed privacy-preserving computation as a potential opportunity for bolstering nuclear safeguards, this chapter illustrates how these types of calculations might be implemented. It describes two simulations that were designed and performed to find anomalous events—potential material diversions—in data that are highly relevant for drawing safeguards conclusions. Like in a real safeguards scenario, each of the demonstrations discussed in this chapter replicates the interaction between a nuclear facility and a regulator. Since the calculations are privacy-preserving, the data supplied by the facility are kept secure and never explicitly revealed to the inspecting party. The goal here is to demonstrate as a proof-of-concept that privacy-preserving computations can be performed on radiation spectra, pushing the current boundaries of nuclear safeguards capabilities.

To date, data analytic solutions to safeguards challenges are only conducted with data collected at a facility that is then transmitted to the International Atomic Energy Agency (IAEA) under the assumption that the IAEA will maintain the confidentiality of that data [42]. Privacy-preserving techniques are not included in any traditional safeguards systems, and this work represents the first attempt to prove their viability. Even beyond the field of nuclear safeguards, the computational intensity of multiparty computation (MPC) calculations has resulted in few attempts being made to apply MPC to data analytics solutions that operate on time series datasets, especially those outside niche problem areas. While this study only directly demonstrates the applications of MPC to nuclear safeguards datasets, the concept of conducting privacy-preserving anomaly detection offers a novel approach to monitoring and surveillance practices.

Before covering the anomaly detection algorithms and privacy-preserving calculations of those two simulations in detail, the first section of this chapter will introduce the safeguards relevant dataset that was used for the analyses. This dataset consists of gamma-ray spectra collected over the course of two months at Oak Ridge National Laboratory (ORNL). Along with a description of the experimental setup and an overview of the data format, this introduction will also explain how the two trials were structured in order to make use of the information contained in the dataset. Once this basic understanding of the dataset driving the simulations has been established, the next section will review the basics of time series anomaly detection. This discussion will build the necessary foundations for the algorithmic design and deployment explanations to follow. Finally,

the last two sections of this chapter will cover the privacy-preserving simulations that were actually performed. These sections will include a complete description of both the chosen anomaly detection algorithms, as well as details about the garbled circuits prepared using Obliv-C, a software library for performing MPC.

4.1 Modeling a Nuclear Safeguards Datastream

As discussed in Chapter 2, no type of data is more important to monitoring nuclear material than radiation measurements, especially gamma-ray spectra. Bearing that in mind, along with an understanding that stronger demonstrations of viability would be more likely to encourage future adoption of MPC anomaly detection in safeguards systems, this work placed paramount importance on gaining access to a high-quality set of radiation spectra.

Still, high-quality time series of radiation spectra are difficult to acquire for a myriad of reasons. First and foremost, the experiments collecting such datasets face a litany of physical and technical challenges associated with their operation: acquiring the necessary radioactive sources, calibrating detectors, complying with necessary rules and regulations surrounding the handling of nuclear material, and compiling the data in an organized and consistent format. These datasets also tend to be quite large, and so assembling them requires substantial effort on the part of highly specialized researchers. While any given radiation spectrum is unlikely to be particularly large in isolation, a time series of radiation spectra can grow in size rapidly. With hundreds of channels, and every channel providing an integer value of counts, a single radiation spectrum can potentially reach the order of tens of kilobytes. Together, a collection of spectra can then easily produce a multi-gigabyte time series for even moderate frequency measurements (e.g. 1 Hz) taken over the course of several days. Although storing datasets of this size is not particularly challenging, hosting public repositories of information on this scale was not particularly common until recently. Then, in addition to any standard data cleaning steps, datasets that could potentially include sensitive or proprietary information may require prescreening to ensure that sensitive information is not shared. Alternatively, sensitive datasets may be placed under access limitations by controlling agencies or businesses, another barrier preventing widespread dissemination of radiation data sources. Finally, even if a research team were technically able to negotiate all of these potential challenges, it is quite possible the effort would not be deemed worthwhile when operating within budgetary constraints and with potentially small target audiences.

Considering the limited availability of comprehensive radiation spectra time series datasets, it is notable that this project was able to access a significant quantity of radiation data, including measurements of material transfers of special nuclear material (SNM). Since SNM constitutes the primary explosive nuclear material in a nuclear weapon, it is likely that such movements would resemble activities of particular interest to a safeguards administrator. The data referenced here was provided by the Modeling Urban Scenarios and Experiments (MUSE) project based out of ORNL, and shared via the Multi-Informatics for Nuclear Operations Scenarios (MINOS) program. The MUSE dataset represented a substantial and continuous collection of radiation spectra, and so serves as a near ideal dataset for performing spectral time series anomaly detection. Although the

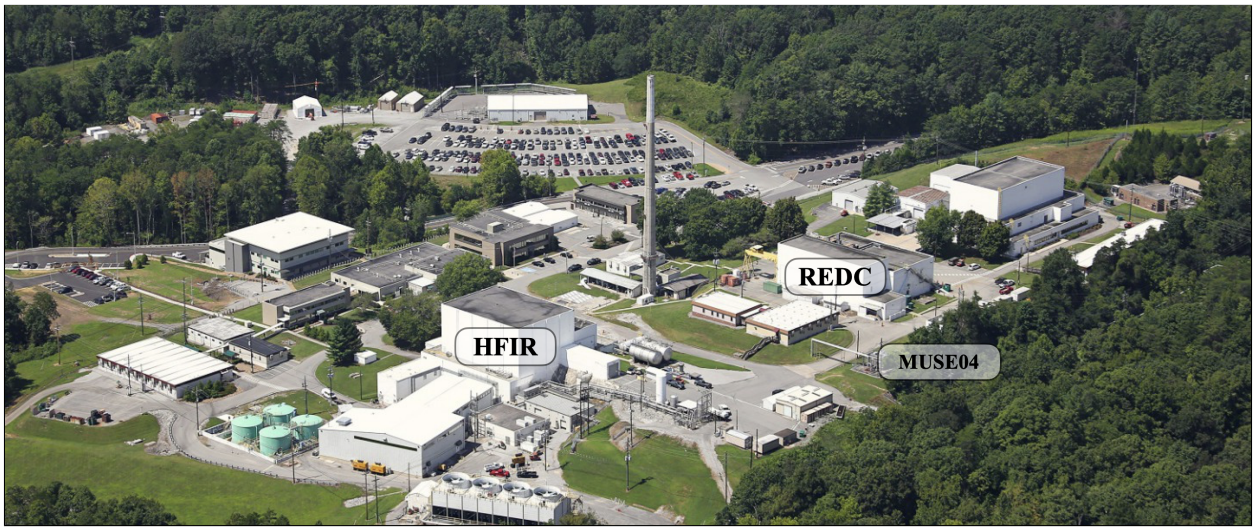


Figure 4.1: The HFIR and REDC facility area at ORNL [102]. The MUSE detectors were located at intervals along the connecting roadways (the MUSE04 detector has its location labeled).

dataset had been rigorously studied by the collection team and essentially all potential anomalies had been identified, the material transfers of SNM produced distinguishable, relatively infrequent signatures that served as a proxy for truly anomalous events. In fact, since these material transfers were recorded on ground truth logs maintained by the data collection team, they proved to be ideal benchmarks for verifying the effectiveness of the privacy-preserving anomaly detection algorithm.

Using the dataset as a model of reality, it became possible to mimic the types of analyses that might prove useful to nuclear regulators using MPC. Details of the two trials illustrating this application of privacy-preserving computation are described later in Sections 4.4 and 4.5.

4.1.1 The MUSE Experiment

The MUSE project's experimental setup was designed to capture radiation signatures arising from the routine daily operations around two facilities at ORNL: the High Flux Isotope Reactor (HFIR) research reactor and the Radiochemical Engineering Development Center (REDC) target processing facility. With unclassified operations involving actinides and routine material transfers (as well as the accompanying activity logs accessible to the data collectors), this area is well-suited for performing facility monitoring of this nature [101]. An aerial photograph of the area, with the HFIR and REDC facilities highlighted, is shown in Figure 4.1.

The experimental arrangement specifically consisted of six MUSE nodes placed along stretches of road surrounding the two facilities. Each of the six nodes contained a set of NaI(Tl) gamma ray detectors and video cameras, and some of the nodes also possessed lidar units. With continuous, autonomous operation, these detectors are similar to many of the unattended monitoring systems (UMSs) that are deployed by the IAEA (see Section 2.3.2). Even after a preprocessing stage, where

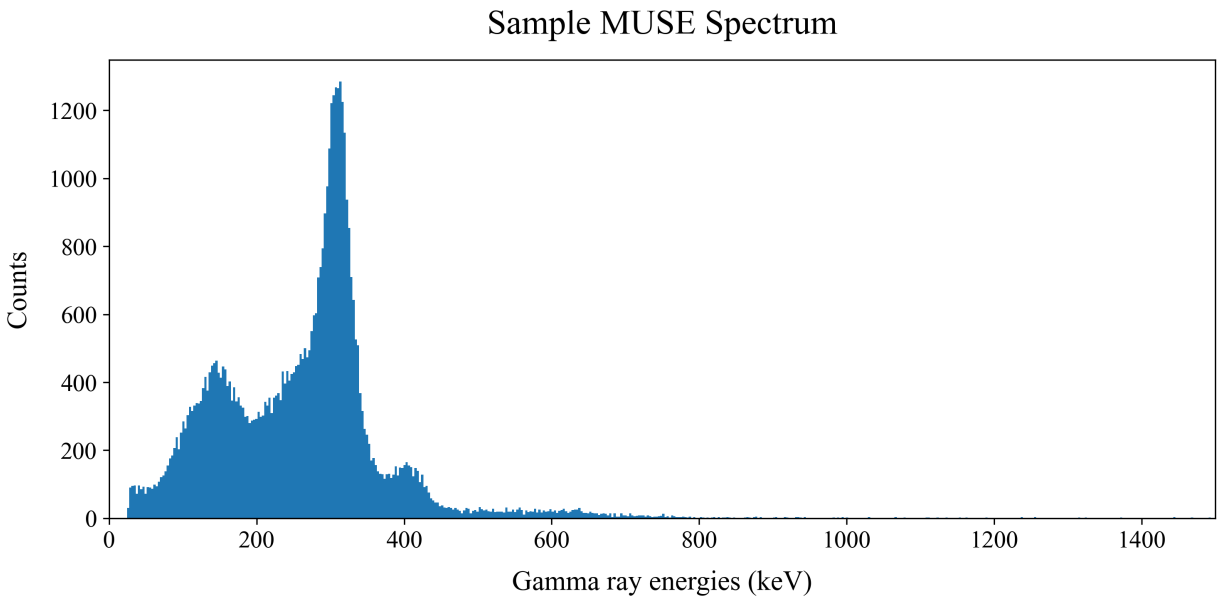


Figure 4.2: A truncated gamma-ray spectrum from the MUSE dataset, with each measured channel representing counts in 3 keV increments. The spectrum is dominated by the 311.9 keV peak generated by the beta decay of protactinium-233 into uranium-233 (and near-immediate de-excitation).

the MUSE team converted the raw dataset sampled at a rate of 10 Hz into an energy-calibrated dataset at 1 Hz, the detectors were able to capture detailed accounts of the area's activity. This included both movements of radioactive material between the two facilities *and* otherwise unrelated background events. These background events were caused by weather phenomena and routine non-nuclear activities associated with lab personnel, in addition to the standard unavoidable levels of environmental background radiation.

4.1.2 The MUSE Dataset

The primary dataset provided for analysis consisted of gamma-ray spectra collected by each of the six detectors between February and March 2019. In addition to those measurements, the accompanying ground truth logs of local activity were provided for comparison, enabling the validation of the chosen anomaly detection algorithms.

For the primary dataset, each of the six 2 in. $\times$ 4 in. $\times$ 16 in. NaI(Tl) detectors measured spectra in 1000 energy channels, with every channel counting gamma rays measured in a 3 keV energy window. Since high-energy gamma rays were considered unimportant for the anomaly detection methods presented in the remainder of this work, only the lowest 500 channels were used in all of the subsequent analyses. For illustrative purposes, Figure 4.2 shows a sample gamma-ray spectrum from the MUSE dataset when material of interest is passing the detector. While the limited resolu-

tion of the NaI(Tl) detectors hinders high-fidelity spectral analysis, it is still possible to distinguish certain high energy peaks that suggest the presence of SNM.

As mentioned in Section 4.1.1, although each of these raw measurements was collected at 10 Hz, the recorded dataset aggregated gamma-ray counts into one second time frames. For each detector node, these 1 Hz measurements were packaged in a series of HDF5-formatted data files, including the 1000 channel spectra, a measurement timestamp, and the corresponding detector dead-time corrected live-time for the interval. In total, the complete dataset comprises several thousand HDF5 files, each containing approximately four-hours of data collection.

Accompanying this primary dataset of detector measurements was a set of ground truth logs that described the most notable material transfer events taking place in the vicinity of the nodes. These logs include information on the event dates, the items being transferred and their material compositions, the shipping containers, and the origination and destination points of the transfers. Although the limited resolution of the NaI(Tl) detectors likely obscured some of the potential spectral features that would have arisen from the circumstances described in the logs, the available information was still sufficient for confirming the dates, approximate contents, and travel paths of SNM through the detector setup.

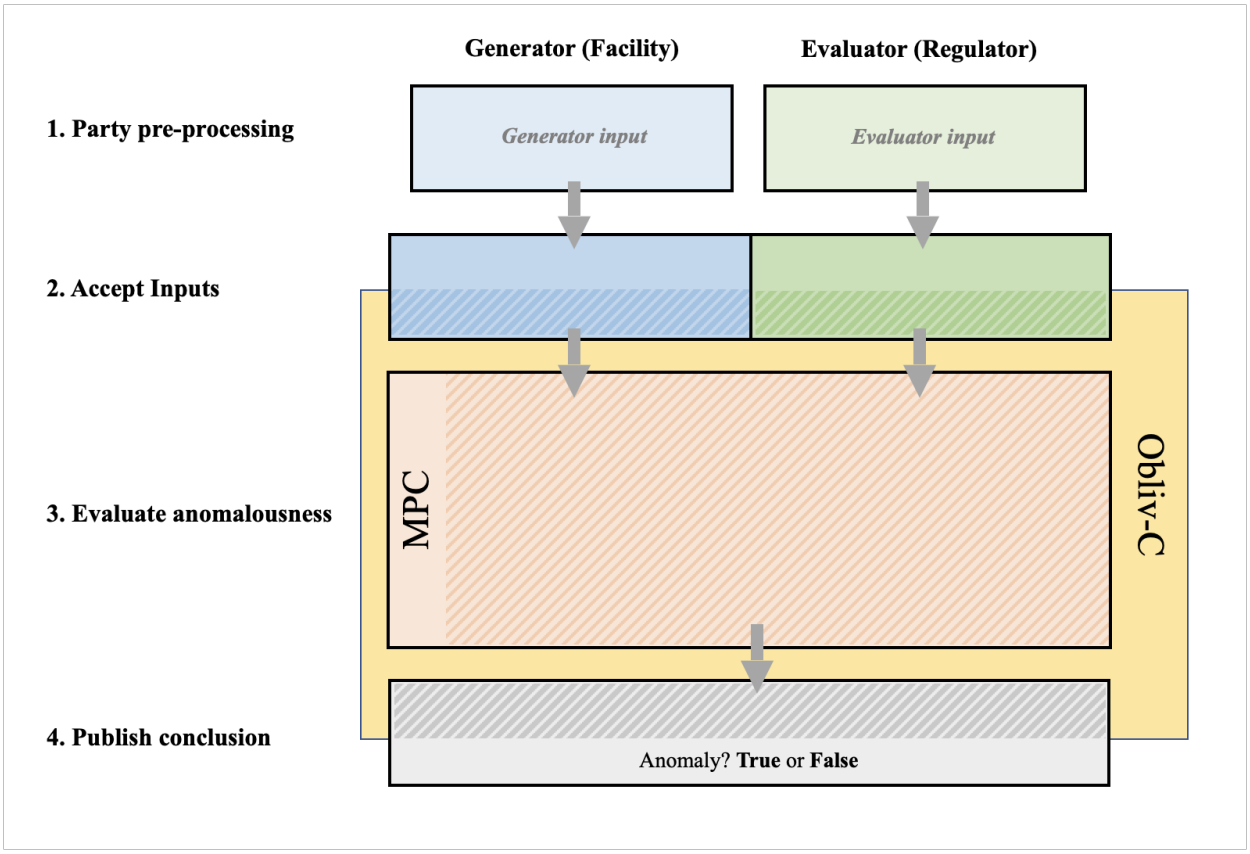
4.2 A Computational Experiment

With access to the MUSE dataset, two privacy-preserving anomaly detection algorithms were developed to operate on that dataset, and the two trials testing the efficacy of those algorithms are presented in Sections 4.4 and 4.5. While those two sections provide a comprehensive description of each algorithm, both methods were selected due to their compatibility with the garbled circuit constructions provided by the Obliv-C framework. Specifically, these were anomaly detection algorithms that could be represented relatively efficiently in circuit-format—those that minimized circuit size, reduced reliance on conditional statements (especially when used recursively), and avoided frequent computationally intensive recalculations as more data became available.

In general, the two trials demonstrating the privacy-preserving implementations of these anomaly detection techniques were organizationally similar. An overview of the shared trial procedure is presented as a template in Algorithm 1. Specific details for each step are specified below and the template will be filled in for each case. Two parties were simulated by the developers for each trial: one party representing a nuclear facility under safeguards and another representing a safeguards administrator. Within this model, a garbled circuit was prepared using Obliv-C to implement an MPC version of an anomaly detection algorithm. Obliv-C and the garbled circuit preparation process are described in detail in Section 4.2. This algorithm was then applied to the measured data from one of the six MUSE detectors, which served as the private input of the party adopting the role of a nuclear facility (the circuit generator, in the garbled circuit protocol). The inspecting party (the circuit evaluator) provided algorithmic parameters as its inputs. In both cases, the simulated nuclear facility was the only participant to ever have access to the entirety of the MUSE dataset.

For the first trial, just the MUSE data from the month of February was analyzed by the garbled circuit anomaly detector. Playing the role of a safeguards administrator, the procedure sought to

Algorithm 1 A sketch of the procedure followed by each of the two trials presented in Sections 4.4 and 4.5.



1. Both parties compute their input data in a pre-processing phase as necessary.
2. Inputs are securely passed into the Obliv-C implementation of the garbled circuit.
3. Inputs are encoded in the garbled circuit of the anomaly detection algorithm (labeled MPC in the diagram).
4. The circuit is evaluated and any detected anomalies are revealed to the participants.

locate the date and time of a single material transfer event containing a mixture of neptunium and protactinium that was known to have occurred at some point during the month. The results showed that the algorithms had some success in identifying anomalous behavior correctly while preserving the privacy of the input spectra, with the complete details provided in Section 4.4.

Building on the results of the first trial, the second trial advanced to use a more sophisticated algorithm, described thoroughly in Section 4.5. This trial relied on data from the complete two month interval spanned in the dataset, using the data from the month of February as a training dataset for a program analyzing the March data for anomalies. It was known that there were several potential material transfer events taking place during the month of March; however, unlike in the first trial where it was known that only one event transpired, the party playing the role of a regulator was not privy to any *a priori* information about the dates, times, frequency, or nature of the events during that time period. Instead, this second trial actually leveraged information learned during the first trial to improve its accuracy.

Recognizing that nuclear inspectors in the field would be limited in the amount of information at their disposal when developing MPC-based safeguards systems, this study correspondingly placed limits on the amount of information that was available to the system developers. This meant that beyond just siloing the data inputs of both participants, the study actually attempted to imitate the real-world conditions faced by safeguards administrators by restricting the access to the ground truth data logs. It was only after the first trial's analysis was complete that the developers were able to access the ground truth logs for the month of February to validate that algorithm's findings. The second trial proceeded in like manner, with developers having no access to the March ground truth logs until after the March data had been (securely) assessed and potential anomalies identified.

Building Garbled Circuits with Obliv-C

Both of the test trials relied on Obliv-C as the privacy-preserving engine driving the MPC implementation of each anomaly detection algorithm. The Obliv-C framework is a strict extension of the C programming language, and allows programs to be constructed using a C-like syntax that is then compiled into an equivalent garbled circuit [99].

The development procedure of garbled circuits using Obliv-C is designed to be straightforward for general users. Using elements provided by the Obliv-C library, a developer constructs a protocol description for the chosen algorithm. The Obliv-C compiler then translates the protocol description into a version of Yao's garbled circuit protocol. Every protocol description accepts private input from two parties, and can be written to return private, jointly computed outputs that must be intentionally revealed by the compiled program to maintain security.

Along with traditional C keywords and logic structures, the Obliv-C library provides a variety of its own `obliv` qualified data types and logic structures. The `obliv` qualified data types store private inputs and any quantities derived from them, as well as guide the compiler in constructing secure operations using those data elements. Beyond those `obliv` qualified types, the language also provides the ability to write functions and conditional statements that operate on these datatypes in a secure way. Such structures allow secure implementations of standard C-programs to be constructed in a style familiar to software developers, although they may require some additional effort during the

development process. For example, the `obliv if` logic structure allows conditional statements to be evaluated based on privately evaluated expressions, and the `if/else` pattern is a common feature of nearly every modern programming language. However, in general, executing code for either leg of a conditional (true or false) reveals information about the evaluated value of the condition. To avoid revealing this information and maintain the privacy of those conditional values, the `obliv if` block is designed to build and simulate *both* legs of the conditional statement. It is left to the decryption process to ensure that only assignments made on the "correct" conditional leg persist through the calculation and impact the final output, while the evaluator never learns which path through the conditional the algorithm ultimately follows. This dual-execution of conditional statements is highly unusual, restricting a programmer's ability to assign variables that are not `obliv` qualified during a conditional block or use recursive functions that terminate based on a condition.

After compilation, the Obliv-C garbled circuit is produced as a binary executable that may be run by both parties engaged in a computation. The executable is designed to operate over a Transmission Control Protocol (TCP) network connecting the two parties engaged in the calculation over the internet. In this work, where both parties engaged in all of the privacy-preserving calculations were simulated, the protocol was only run locally; however, configuring the executable to run between remote parties would be preferable in almost any safeguards context and this work could easily be repeated with that setup.

4.3 Foundations of Time Series Anomaly Detection

In order to operate without human intervention, sophisticated remote monitoring systems require anomaly detection algorithms to make determinations based only on data collected by the system. Reliance on automated procedures becomes doubly important for monitoring systems that employ privacy-preserving protocols since those protocols require secure programs to evaluate joint computations. In the general outline of the privacy-preserving trials designed here, depicted in Algorithm 1, the anomaly detection technique is the primary component of the third step, evaluating abnormality.

To effectively formulate an anomaly detection process as an algorithm, the notion of a time series anomaly must be precisely defined. Considering a time series anomaly to be a point (or set of points) constituting an outlier among the time series dataset, the intuitive definition proposed by Hawkins is generally appropriate. That definition posits that an outlier is "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [103].

It is worthwhile to note here that not all outliers in a nuclear safeguards time series would necessarily represent anomalous events of interest to safeguards administrators, who would be primarily concerned with diversions of direct or indirect use nuclear material. A distinction is commonly drawn between two types of outliers that meet Hawkins's definition: the first type are outliers constituting anomalous events of interest in a time series; the second are outliers representing unwanted data that in some sense contaminates the dataset. In many contexts, the former is considered to be

a true anomaly while the latter is deemed to be noise [104].[1] Distinguishing outliers in this way must be done on a case-by-case basis, depending on the nature of the problem. This is especially true for nuclear safeguards datasets, where radiation spectra are measurements of highly statistical processes, and so noise-based outliers are unavoidable.

To characterize the wide variety of anomaly detection scenarios, Blázquez-García et al. [106] extend the conventional definition of an outlier and create a taxonomy for recognizing and classifying outliers and outlier identification techniques. The taxonomy features three axes based on the underlying time series type (univariate or multivariate), the analysis method (univariate or multivariate), and the type of outlier being identified (point, subsequence, or time series outliers).

Considering the first axis, time series are designated as one of two types: univariate or multivariate. Univariate time series represent the more traditional concepts of a time series dataset, consisting of an ordered set of real-valued observations over some time interval. Mathematically, the time series can be defined as

$$X = \{x_t \mid t \in \mathcal{T}\} \ , \tag{4.1}$$

a set of observations x_t collected at time $t \in \mathcal{T}$. Formally, each time t may be considered a time index mapping to a real time, such that the complete set of time indices in the time series are given by non-negative integers, $\mathcal{T} \subseteq \mathbb{Z}^+$.[2] Each observation x_t denotes the realization of some random variable X_t. Common examples of univariate time series would include temperature readings at a given location for each hour over the course of one year, or an electrocardiogram heartbeat measurement of voltages every millisecond for five minutes. In the context of nuclear safeguards, measurements recorded by neutron counters (such as those mentioned in Section 2.3.2) would be univariate time series, as these detectors record one neutron count measurement x_t at each time t.

The definition of multivariate time series extends the univariate definition for measurements collected in N dimensions. For multivariate time series, the ordered set of real-valued observations is

$$\boldsymbol{X} = \{\boldsymbol{x}_t \mid t \in \mathcal{T}\} \ , \tag{4.2}$$

where $\boldsymbol{x}_t = (x_{1t}, \ldots, x_{Nt})$ is a N-dimensional vector of observations collected at time t, where now each measured value x_{kt} is a realization of random variable $X_{kt} \in \boldsymbol{X}_t = (X_{1t}, \ldots, X_{Nt})$. Hourly weather readings combining temperature, pressure, precipitation, and/or humidity or a single patient's electrocardiogram consisting of at least two periodically measured voltage signals would be instances of multivariate time series. Multivariate time series are of particular importance in the realm of nuclear safeguards, as gamma-ray spectrometry can be considered a multivariate measurement technique with dimensionality N equal to the number of measured energy channels.

While input data may be either univariate or multivariate, the second axis of the taxonomy considers the analysis methods. These methods may also be either univariate or multivariate in nature. Univariate analysis methods make use of only one variable in identifying anomalies, while multivariate methods consider some or all of the variables in question. It should be noted that these

[1] For more rigorous mathematical classification of these designations—formally strong and weak outliers—see Knorr and Ng [105].

[2] Although seemingly contrived, this notation allows continuous time series intervals to be expressed cleanly in set notation, even when sampled irregularly.

methods are not necessarily restricted to the corresponding input data types; since univariate analysis techniques consider only a single variable, they may be applied to either univariate input data or a single isolated variable in a multivariate time series [106].[3]

Separate from classifying data types and analysis techniques, the third axis of the taxonomy distinguishes types of outliers that may be detected. It classifies outliers as being one of three types: a point outlier, a subsequence outlier, or a time series outlier. The point outlier category is the simplest, where a single anomalous point at some time t is identified as the outlier—either the point x_t (in the univariate case) or $\boldsymbol{x}_t$ (in the multivariate case). A common interpretation of a point outlier is any single point in the time series that deviates significantly from its expected value, with a significant deviation being determined using some threshold τ. In the most general case, such a relationship can be formulated using some metric function f such that a measurement x_t is classified as anomalous if it satisfies the equation

$$f\left(x_t, \overline{x_t}\right) > \tau \ . \tag{4.3}$$

For point outliers, two distinct models exist for determining $\overline{x_t}$: *estimation models* and *prediction models* [106]. Estimation models consider complete time series in their entirety after they have been collected, using all of the time series data points to generate an estimate $\overline{x_t}$ for the true expected value of random variable X_t. Often, this expected value is taken to be $\overline{x_t} = \mathrm{E}\left[X_t\right]$, the mean value of many independent realizations of X_t. Such estimation models are generally useful for making a determination of whether any given point in a historical record might be an outlier. Prediction models, on the other hand, do not have access to the complete set of time series data. Instead, they only have access to the data collected up to the current point in time. The data available to the algorithm is used to establish a prediction of the expected value at the next timestep, and then the measured value x_t is compared against the prediction of the expected value $\overline{x_t}$ and the threshold τ to classify the point as an outlier or not. For nuclear safeguards applications where on-line monitoring components are a key feature, prediction models are particularly attractive. By looking at data collected prior to the present, prediction models generate hypotheses about future data that can be used for analyzing data-streams in real-time. Because of this, both trials described later in this chapter rely exclusively on prediction models, though estimation models would work equally well in cases where a complete time series is available.

Using a simple prediction model to perform point outlier detection on a univariate time series is straightforward. Like in the case of estimation models, the estimated expected value $\overline{x_t}$ is most often found using a traditional approach—for example, as the mean of all previously acquired data or some subset of it. Other models use metrics other than the mean, instead considering the "expected value" to be more accurately represented by a median or another more sophisticated statistic [107, 108]. When the mean is used as an estimate of the expected value, the data from which it is calculated effectively constitute a set of training data for the predictive model. In cases where the complete set of prior data may not serve as a good predictor of future behavior, this training dataset may be limited to only a window of recently collected data points, generating a rolling average. Other times, to avoid needing to recompute the mean with each new datapoint, the predictive

[3] Multivariate analysis techniques may not be applied to univariate time series, however, since additional dimensions cannot be fabricated for those datasets.

model may be "trained" with a fixed subset of the data deemed to be representative. In either case, to generate an accurate prediction using the mean as a metric, the training data for the model ought to be free of anomalies.[4] In the field of nuclear detection and nonproliferation, such anomaly-free collections of radiation spectra are designated as benign source populations (BSPs) [109].

With the expected value defined, determining whether a measurement represents a significant deviation from the expected value may be as simple as finding the absolute difference of the measured point from the expectation. For univariate time series, this is

$$|x_t - \overline{x_t}| > \tau \ , \tag{4.4}$$

where τ is commonly set to be some number of standard deviations away from $\overline{x_t}$.

For multivariate point outliers, Equation 4.4 may be generalized to k dimensions, although a metric other than the absolute value is required for comparing the multivariate data points to a scalar threshold τ. The Euclidean distance,

$$\sqrt{(\boldsymbol{x}_t - \overline{\boldsymbol{x}_t})^\intercal \, (\boldsymbol{x}_t - \overline{\boldsymbol{x}_t})} > \tau \ , \tag{4.5}$$

is one such multidimensional metric, although it is oblivious to variations in the dimensional distributions. Other metrics are able to account for these differences, like the Mahalanobis distance:

$$\sqrt{(\boldsymbol{x}_t - \overline{\boldsymbol{x}_t})^\intercal \, \boldsymbol{S}^{-1} \, (\boldsymbol{x}_t - \overline{\boldsymbol{x}_t})} > \tau \ . \tag{4.6}$$

Here $\boldsymbol{S}$ defines the covariance matrix of the set of observations, namely the time series $\boldsymbol{X}$ (or the subset thereof used to determine $\overline{\boldsymbol{x}}_t$). Of course, in the case that X_t is drawn from a specific known distribution, the threshold τ may be set in relation to that distribution.

There are advantages to identifying other types of outliers besides point outliers, however, and privacy-preserving algorithms could make use of them instead. Rather than being an outlier of just a single point, an anomaly may consist of several points that are considered anomalous when taken together. Analyzing several points together may enable anomalies to be distinguished in intervals of activity where no single point is out of the ordinary but activity patterns are abnormal.

One popular class of anomaly detection techniques that finds anomalous trends rather than points searches for subsequence outliers. As subsets of full time series, subsequences may be defined to reflect the original time series definitions presented in Equations 4.1 and 4.2. Specifically, a length $s \leq |\mathcal{T}|$ univariate subsequence starting at time t would be defined as the continuous set of points

$$\tilde{X} = \{x_t, x_{t+1}, \ldots x_{t+s-1}\} \ , \tag{4.7}$$

while an otherwise equivalent multivariate subsequence would be described as the collection of multivariate data points

$$\tilde{\boldsymbol{X}} = \{\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_{t+s-1}\} \ . \tag{4.8}$$

[4] If, instead, the prediction model metric were to be based on supervised machine-learning, including known anomalies in the training data would be essential. Even other metrics, like the median, are more tolerant of outliers being included in the training data.

In both cases, the subsequence starting time t must be chosen so that the subsequence does not exceed the bounds of the time series; it must obey the constraint

$$t \leq |\mathcal{T}| - s + 1 \,. \tag{4.9}$$

Alternatively, instead of analyzing a subsequence of the time series, an algorithm could evaluate whether one of the variables in a multivariate time series behaved anomalously over the duration of the time series. While the subsequence methods isolate time periods as anomalous or not, this "time series outlier" method categorizes individual variables as being anomalous. In practice, this methodology is far less commonly studied than subsequence anomaly detection [106].

The choice of which anomaly detection algorithm to use is often tightly coupled to the goals and requirements of the detection algorithm as well as the types of input data to be analyzed. This is especially true when considering nuclear safeguards scenarios, where a regulator may only have access to certain modalities of data. For instance, when applying anomaly detection algorithms to the output of neutron counters that are designed to monitor for movements of SNM, the input data are univariate and therefore a univariate method must be chosen. On the other hand, when inspecting gamma-ray spectra, the input data are multivariate and so multivariate techniques may offer greater power for distinguishing anomalies. Multivariate methods are usually needed when anomalies are generated by a certain phenomenon, such as a specific radioactive isotope of interest, that may be identified by recognizing the relationship of dependent variables (e.g. energy channels). There are, however, situations where univariate techniques might be advantageous for even multivariate nuclear safeguards datasets. Since they are generally simpler, univariate analyses may be preferable when working on systems with limited computational resources, or, critically for this work, when those calculations are performed via computationally expensive privacy-preserving algorithms.

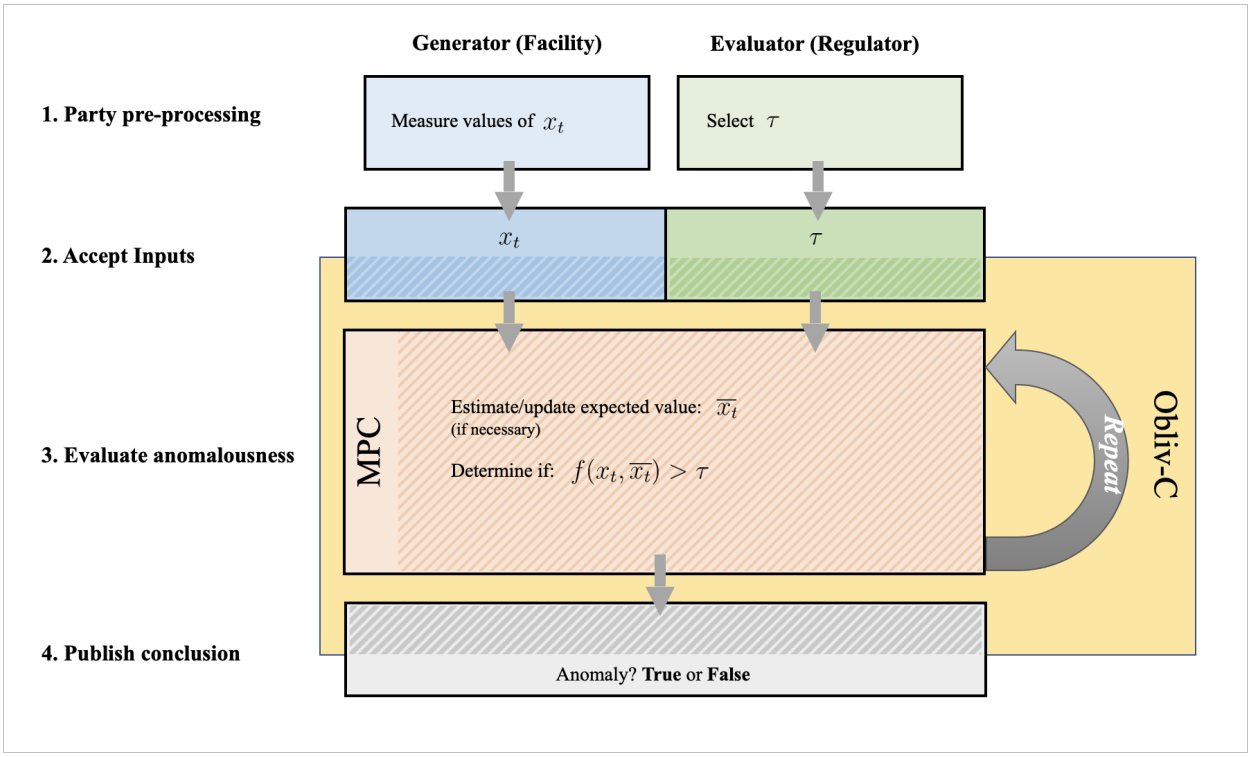
Among the wide variety of anomaly detection algorithms, including those reviewed in Section 4.3, there are many candidates that could be successfully applied to a time series of radiation spectra while still being practically implemented via MPC. Due to the simplicity of point outlier detection, the garbled circuits that have been created to locate and identify anomalies in the MUSE dataset rely primarily on techniques that search for point outliers. With that in mind, it is possible to update the algorithmic sketch presented earlier with details specific to point outlier detection methods; the revised diagram is presented in Algorithm 2. Equation 4.3 represents the objective of the MPC calculation when x_t and τ represent private inputs provided by the circuit generator and circuit evaluator respectively.

The remainder of this chapter offers a case-study in applying two types of anomaly detection algorithms to time series of gamma-ray spectra and quantities derived from them, implemented in a privacy-preserving manner.

4.4 Trial 1: Gross Count Anomaly Detection

The first privacy-preserving anomaly detection trial was designed to discriminate anomalous events in the MUSE dataset's collection of February spectra using the gross total of gamma rays detected at each point in time. Having determined that point outliers represented the best category of outlier

Algorithm 2 An updated outline of the procedure used in each of the two trials, reflecting the fact that both trials attempt to detect point outliers.



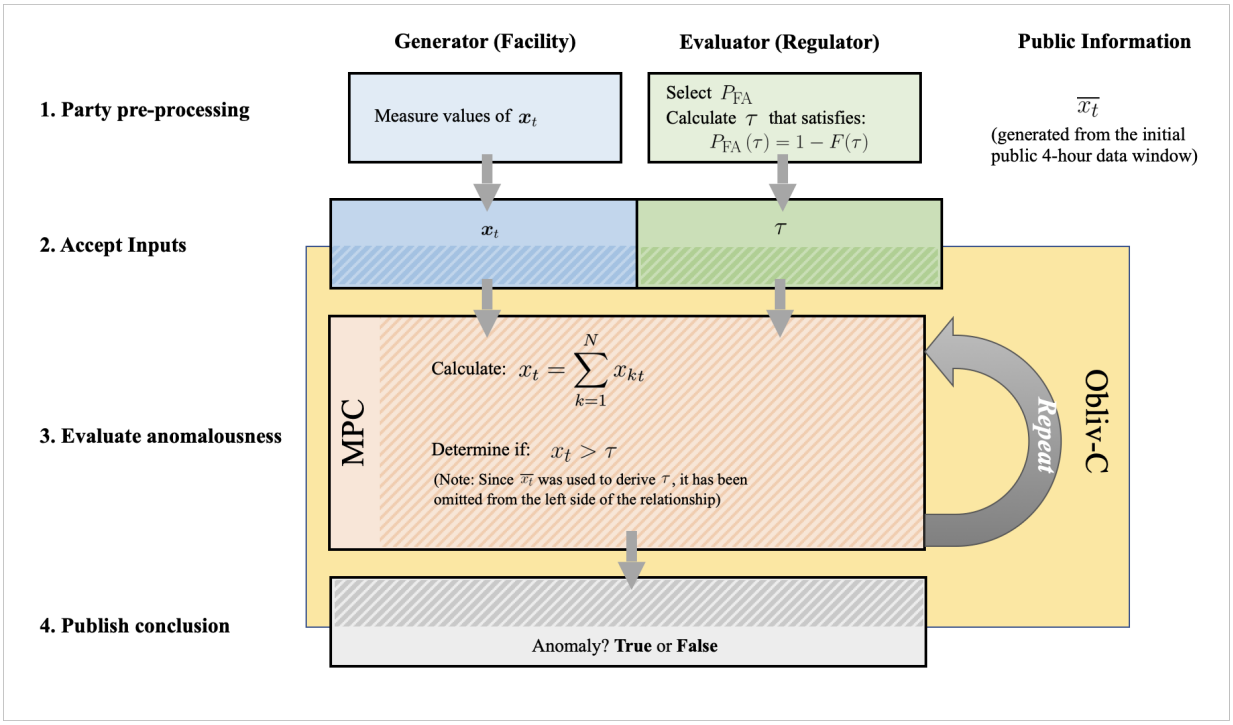
to use for identifying anomalies, it remained to choose either a univariate or multivariate method to perform the anomaly detection procedure. Noting that radioactivity measurements fluctuate randomly and each spectrum covers several hundred channels, it was logical to select a multivariate method for detecting point outliers in multivariate data.

For multivariate spectral inputs collected over the set of times $\mathcal{T}$, each point $\boldsymbol{x}_t$ would be a spectrum measured at time $t \in \mathcal{T}$. Each spectrum would contain a number of measured counts in each of the N spectral channels, expressed as $\boldsymbol{x}_t = (x_{1t}, \ldots, x_{Nt})$. However, by recognizing that the gross number of counts measured by a detector is a common value of interest to nuclear safeguards analysts, the anomaly detection algorithm can be simplified further. The next section describes the gross counting algorithm's procedure, followed by an explanation of specific implementation details in Section 4.4.2, and results in Section 4.4.3. Because the results of the first implementation of the algorithm suggested an adjustment that would likely improve the gross counting algorithm's accuracy, Section 4.4.4 describes changes to the algorithm and implementation specifics, along with an updated set of results.

4.4.1 The Gross Counting Algorithm

Determining anomalousness using the gross counting method requires that each measured data point (spectrum) $\boldsymbol{x}_t$ be evaluated using an equation of the form of Equation 4.3. That equation leaves the evaluation function f and threshold τ to be determined, and so this section explains the choices used to implement the gross counting algorithm. It is this algorithm that is then expressed as an Obliv-C protocol description and compiled into a garbled circuit, as was described in Section 4.2. Algorithm 3 illustrates how each element of the algorithm fits into the complete MPC implementation.

Algorithm 3 A diagram describing the privacy-preserving implementation of the gross counts anomaly detection algorithm used in the first trial.



By aggregating the counts in each channel, the multivariate spectral time series produced by each detector is effectively transformed into a univariate time series. In this case, each time series point is given by x_t, a simple sum over the detector's N channels:

$$x_t = \sum_{k=1}^{N} x_{kt} \, . \tag{4.10}$$

Having represented the multivariate time series by a univariate one, the common interpretation of a univariate point outlier discussed in Section 4.3 can be applied; x_t is classified as anomalous if it exceeds some threshold τ.

To perform this classification, τ must be determined. Radioactive decay behaves as a random process, and so there are no physically fixed values of τ that would absolutely indicate an anomalous event. Instead, the statistical predictability of the decay events suggests that τ should be selected using statistical arguments. Ideally, this would be a value of τ large enough to reliably detect anomalies while yielding only an acceptably low rate of false positive events. This false positive rate may be chosen based on the requirements of the application and, once selected, can be used to derive back an appropriate threshold value.

The value of τ corresponding to a given rate of false positive events may be deduced from the probability distribution of detected gamma-ray counts. In this case, the gross count of measured gamma rays meet the criteria for being accurately described by a Poisson distribution: the count is always a non-negative integer, measured radioactive decay events are typically independent of each other, and the average decay rates are effectively constant for short[5] time intervals. Since x_t may be treated as if it were sampled from discrete random variable X_t,[6] the probability that x_t counts are measured at time t is determined by the Poisson distribution probability mass function:

$$f(x_t) = \Pr\left(X_t = x_t\right) = \frac{\mu^{x_t} e^{-\mu}}{x_t!}\,, \quad x_t \in \mathbb{Z}^+\,. \tag{4.11}$$

The parameter μ denotes the expected number of counts to occur during the observation period.

Given such a distribution function, the probability that x_t will be no more than c in any particular observation period can be expressed using the cumulative distribution function (CDF). For a discrete random variable, the CDF of the Poisson distribution function is a summation over the probability mass function:

$$F(c) = e^{-\mu} \sum_{i=0}^{\lfloor c \rfloor} \frac{\mu^i}{i!}\,. \tag{4.12}$$

$F(c)$ represents the probability that the number of gamma rays counted in any observation is c or fewer. Setting the anomaly detection threshold τ equal to c, the natural occurrence of $x_t > \tau$ gamma rays in a complete spectrum would represent a false positive event. As such, the probability that any event is a false positive measuring more than τ gamma rays is given by the survival function evaluated at $c = \tau$:

$$P_{\text{FA}}\left(\tau\right) = 1 - F(\tau)\,. \tag{4.13}$$

Since $P_{\text{FA}}\left(\tau\right)$ is defined by the application requirements, the quantile function of the CDF inverts Equation 4.13 and can be used to determine τ from the desired false alarm probability.

From here, only the Poisson parameter μ remains to be defined for τ to be calculated directly from the false alarm probability. Given that μ represents the expected value of the Poisson variable X_t,

[5]This is assuming that isotopes with characteristic lifetimes on the order of a minute or less have largely decayed before being transported in front of the MUSE detector, rendering the activities of the remaining isotopes to be relatively constant over time periods on the order of seconds.

[6]Note that the Poisson nature of the measured counts generally holds over any range of energies. Recall that x_t is technically the sum of measurements x_{j_t} for each of the k detector channels, but these may each be treated as if they were sampled from k Poisson-distributed discrete random variables X_{j_t}.

it can be conveniently approximated using the experimentally calculated expected value $\overline{x_t}$. To be a valid expected value in the Poisson distribution, $\overline{x_t}$ must be generated from an anomaly-free BSP.

For the first trial in this work, the BSP was initially defined to be a continuous window of data excerpted from the complete time series of MUSE radiation spectra during an interval when it was known that no anomalies (transfer events) had occurred. As a continuous collection of points, the BSP is a subsequence containing s consecutive measurements over an interval starting at some time t_B. Denoting the set of times comprising the BSP as $\mathcal{B}$, that set is explicitly given by

$$\mathcal{B} = \{t_B, t_B + 1, \ldots, t_B + s - 1\} \ . \tag{4.14}$$

Recalling that this trial's algorithm followed a prediction model, the BSP could *only* include data collected prior to the current time t. The starting point of the BSP was therefore subject to a constraint analogous to Equation 4.9:

$$t_B \leq t - s + 1 \ . \tag{4.15}$$

More succinctly, this constraint simply stipulates that there must be ample prior data to make an estimate at time t. Given this definition for a continuous BSP, the expected value of a measurement can be set equivalent to the expected value computed over the BSP and calculated using the standard formula:

$$\overline{x_t} = \frac{1}{s} \sum_{i=t_B}^{t_B+s-1} x_i \ . \tag{4.16}$$

With $\overline{x_t}$ serving as the estimate of μ in the Poisson distribution and a false alarm probability being established by the circuit evaluator, the percent point function for the Poisson distribution (the inverse of the CDF in Equation 4.12) can be used to directly calculate threshold τ.

4.4.2 Identifying MUSE Anomalies Using Gross Gamma Ray Counts

The gross counting algorithm was applied—in the form of a garbled circuit—to the MUSE dataset for just the month of February. Recall that Obliv-C was used as the privacy-preserving engine, with the algorithm being translated into a version of Yao's garbled circuit protocol by the framework's backend, as described in Section 4.2. For the party acting as the nuclear facility, the private input consisted of just the MUSE data. The nuclear regulator had its own private input: the gross count threshold τ for distinguishing anomalies from background and other non-anomalous events.

Although the MUSE experiment collected data from six NaI(Tl) detectors, the data from detector 4 was selected to be the nuclear facility's inputs to the garbled circuit. This detector was the only one placed directly on the material transfer pathway between REDC and HFIR, and so represented the best option for monitoring activity taking place there.

On the other side of the calculation, the party playing the role of the nuclear regulator was responsible for calculating and supplying the anomalous threshold τ as its own private input. First, considering the requirements of a safeguards program, as well as the fact that nearly a complete month of spectral data were available in this trial, the desired false positive rate was chosen to be no more than a single anomalous event per decade. For spectra recorded at 1 Hz, this rate translates to a probability of no more than 3.168×10^{-9} that any spectra is a false positive.

In addition to the false positive rate, calculating τ using Equations 4.12 and 4.13 entails the regulator knowing the expected number of counts per spectrum μ. For the purposes of this trial, estimating this expected value as $\mu = \overline{x_t}$ did, in fact, require some knowledge of the MUSE dataset. Because of this, it was assumed to be public knowledge among both parties that approximately the first four hours of recorded spectra[7] during the month of February were free of anomalies. These spectra were designated as the BSP, and were shared freely between the two parties. While provisioning the data to both parties did diminish the privacy of the nuclear facility's inputs to some degree, the resulting calculation resembled a real world scenario where the regulator would be able to observe an operating nuclear system for some relatively short period of time before transitioning to a privacy-preserving remote monitoring model. Beyond this four hour window, no other spectral data was ever directly shared with the safeguards administrator.

Having public access to the BSP data allowed the inspecting party to calculate a threshold of its choice. The Poisson parameter μ, representing the average number of gross gamma-ray counts recorded per spectrum, could be estimated using the mean value of counts measured in the BSP,

$$\mu = \overline{x_t} = \frac{1}{s} \sum_{i \in \mathcal{B}} x_i \approx 1226 \tag{4.17}$$

(recall $s = |\mathcal{B}|$). Together with the acceptable false positive rate of one anomalous event per decade, this expected value of the per spectrum gross count measurement resulted in a threshold of $\tau = 1435$. Although the value of μ could be calculated by the nuclear facility as well as the regulator (due to the public nature of the BSP spectra), the facility would not be able to learn the target threshold value without knowledge of the regulator's choice of false positive rate.

4.4.3 Preliminary Results

When applied to the February MUSE data, the garbled circuit anomaly detection algorithm was able to privately identify numerous spectra that met the criteria for being anomalous. In all cases, the party assuming the role of safeguards administrator was only ever notified of the time at which an anomaly occurred and never learned either the original spectral data or the aggregated gross count total.

Unfortunately, using the gross counting anomaly detection algorithm set to detect no more than even one false positive anomalous event per decade was found to be too sensitive to be practically useful. Out of the 2.3 million spectra in the dataset, the model identified 142,816 events that it considered anomalous. These events were distributed on all but three days of the month, while it was known that only one transfer event of short duration—on the order of minutes or less—existed in the February data. The false positive rate was much higher in practice than the target rate, which left no ability to make any reasonable conclusion about the true location of the anomaly in the dataset. Even using a notably stricter false alarm rate of only one false alarm detected per century had a marginal effect; when run with that adjustment the algorithm still identified 129,637 events as

[7]The first file (of the 148 provided in the dataset for the month of February) was selected to be the anomaly-free background, and that file contained 14,399 spectra collected at 1 Hz.

anomalies, only about a 9% reduction in identified events. This behavior strongly suggested that the Poisson distribution alone was likely a poor model for the background radiation. Although a fixed set of background radiation sources *would* follow a Poisson distribution, there are other influencing factors that may cause the background radiation to deviate from being reasonably "fixed".

Some potential causes of this deviation are external factors and events that might trigger large scale fluctuations in the background radiation. Weather events or other environmental changes over the month may alter the background baseline subtly, but significantly enough that the original set of training data would no longer be representative of the background measurements later in the month. For example, it is well known in the literature that precipitation triggers a "washout" of short lived radon progeny [110, 111]. When deposited on the ground, this larger-than-average quantity of radioactive material has the ability to shift the baseline background measurement. More specific to the HFIR/REDC complex, the MUSE team has reason to believe small quantities of argon-41 may be systematically released from the HFIR/REDC stack [101]. The relatively high energy (1294 keV) gamma ray produced by the decay of argon-41 could change the background radiation signal measurably, and appears only when HFIR is active—a schedule generally independent of any of the notable transfer events.

Support for the hypothesis that the projected background was not representative of actual background over time can be found in the daily distribution of events over the course of the month. This distribution is plotted in Figure 4.3. It is apparent that the majority of the anomalies occur during the second half of the month (and with increasing frequency), suggesting that the background measured during the first four hours of the month was no longer a realistic characterization of the background occurring later.

4.4.4 Algorithmic Adjustments and Secondary Results

To mitigate the high false detection rate arising from an inaccurate long-term background hypothesis, the algorithm ought to be able to update the background measurement over the course of the data collection period. One solution is implementing a moving average that incorporates new data into the BSP. Updating the BSP in this way effectively retrains the algorithm based on newly acquired information. Several different types of moving averages are potential candidates, including a simple moving average, a cumulative moving average, or a weighted moving average.

Both the simple moving averages and weighted moving averages use a finite rolling window of data points to recompute the average background measurement at each time step [112]. This rolling window only uses the most recent data points in determining the average background, and so either type of method is able to adapt to changes in the background baseline over time—the model effectively "forgets" information about the background that occurred before the window. In many cases, this behavior is reasonable; however, in the safeguards regime, regulators would presumably be averse to having the algorithm totally forget prior information. Such a system may offer more avenues to be exploited, as new information would be tested against only some recent set of data that has been successfully incorporated into the BSP. Valid data from before the BSP window would no longer have any bearing on the current background calculation. Still, these two averaging methods are likely preferable to a cumulative moving average that is calculated simply as the mean of all prior

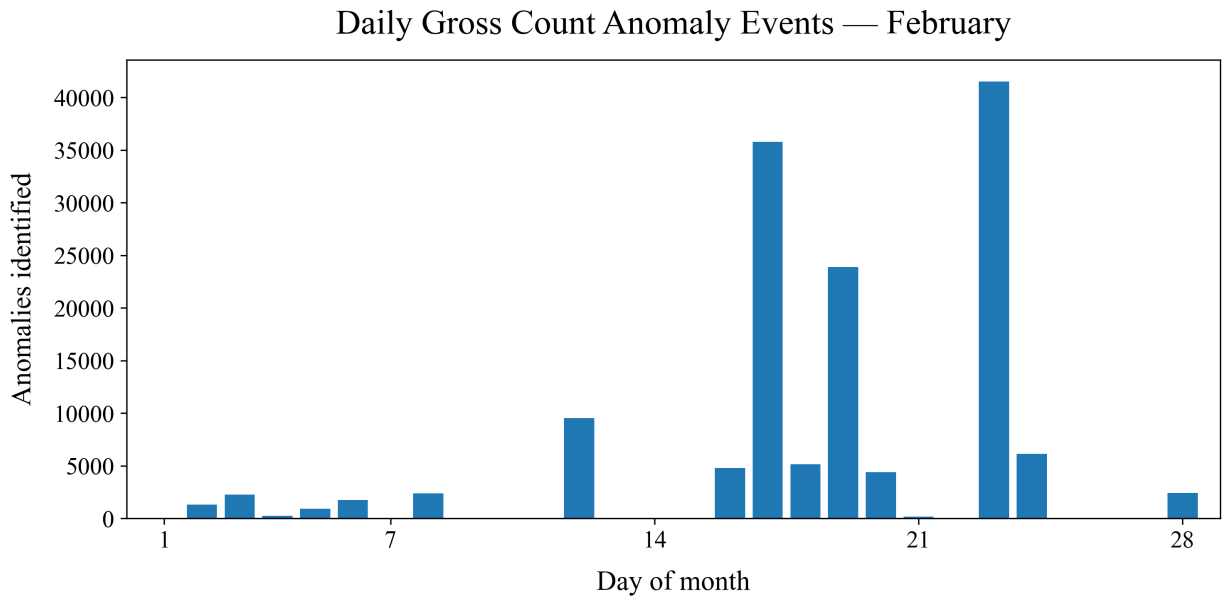


Figure 4.3: Daily events measured over the month of February using a gross counts anomaly detection algorithm and nominal false positive rate of one anomaly per decade. It is evident that the majority of the events identified as anomalous occur in the second half of the month.

data points with no weighting. Such an averaging scheme would consider the dataset's entire history equally when predicting the expected background. In this case, no preference would be given to more recently collected measurements, and a long period of non-anomalous behavior would cause the background to asymptotically approach a constant value. Low frequency shifts in the measured background would eventually cease to have a meaningful impact on the projected background.

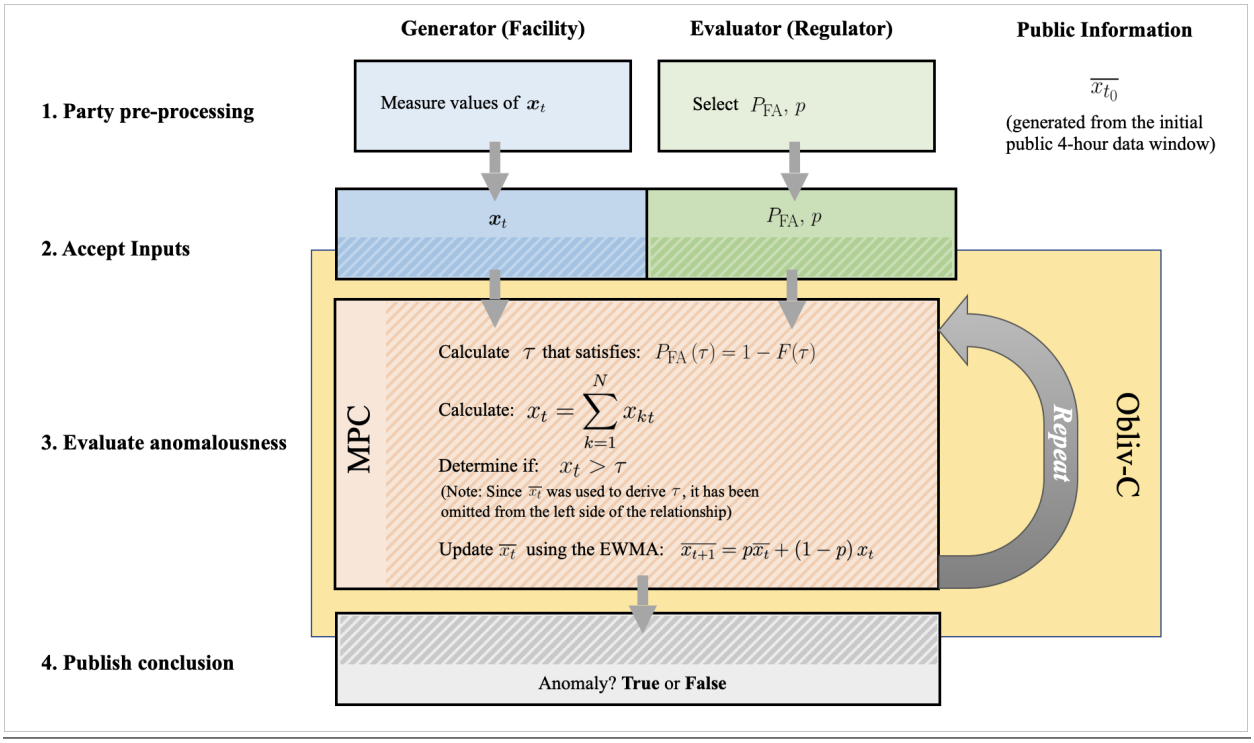
Another alternative, and the one chosen here for adjusting the gross counting algorithm, is an exponentially weighted moving average (EWMA). Like the simple moving average and the weighted moving average, the EWMA provides the strongest consideration towards recently collected data. However, unlike those two window-based averages, the EWMA never completely "forgets" information that falls outside the fixed rolling window. Instead, each time the average background is to be updated, the new gross counting measurement to be incorporated into the BSP is averaged against all of the previously computed background contributions [112]. The exact calculation is given by

$$\overline{x_{t+1}} = p\overline{x_t} + (1-p)\,x_t, \quad 0 \leq p < 1\,, \tag{4.18}$$

where x_t and $\overline{x_t}$ are respectively the gross and expected number of gamma rays measured at time t, and $\overline{x_{t+1}}$ is the projection for the expected number of background counts at the subsequent time $t+1$. The parameter p, termed the *persistence*, represents a weighting factor determining how long older spectra in the BSP contribute meaningfully to the average. Of course, since the BSP must be free of anomalies, the EWMA update procedure is only performed when measurement x_t is not classified

as an anomalous event. This persistence p became an additional private input provided by the party playing the role of nuclear regulator. Algorithm 4 updates the procedure diagram for the privacy-preserving gross-counting algorithm to include the EWMA expansion.

Algorithm 4 The garbled circuit protocol can be enhanced by incorporating an EWMA into the background estimates. Where the expected measurements (background) were previously calculated publicly in advance of the privacy-preserving evalautions, the EWMA required an on-line, secure update of the average using data collected after the initial public window.



Applying the EWMA to the gross counting anomaly detection garbled circuit yielded far more promising results. Table 4.1 highlights the improvement of the anomaly detection algorithm using a BSP determined with EWMA over the algorithm that used a fixed four hour window of data. Out of the more than 2.3 million spectra included in the MUSE February data, now only 55 were flagged as potentially anomalous when using a false alarm rate of one event per decade. Of these 55 anomalous events, 52 were spectra occurring within approximately one minute of 09:19 local time on February 27, 2019. When the stricter false alarm rate of one event per century was used, 46 spectra were flagged as anomalous, *all* of which occurred in the temporal vicinity of the transfer event. Consulting the ground truth logs, this time period coincided with the February material transfer: movement of approximately 650 grams of neptunium irradiation targets, contained in a drum with tungsten shielding, and pushed by hand between REDC and HFIR on February 27. Although the Poisson model may not be perfect, when paired with a BSP calculated using an EWMA, the distribution was adequate for achieving the goal of discriminating anomalous events from evolving background.

Table 4.1: The results of the anomaly detection algorithm for each tested configuration of BSP and false alarm rate.

BSP Method	False Alarm Rate [yr^{-1}]	Anomalies Detected	False Positive Events	
Fixed 4 hour sample	0.1	142,816	142,770	(99.97%)
Fixed 4 hour sample	0.01	129,637	129,591	(99.96%)
EWMA	0.1	52	3	(5.45%)
EWMA	0.01	46	0	(0.0%)

4.5 Trial 2: Region of Interest Anomaly Detection

A potential weakness of the gross counting method lies in the fact that all of the information contained in the spectral shape is ignored by the anomaly detection algorithm. Since anomalies of interest to safeguards administrators are likely to include SNM, algorithms that fail to make use of the entire spectrum leave regulators without a critical analytical tool. Still, given the computational burden of privacy-preserving calculations, many highly sophisticated spectral anomaly detection algorithms may be too complicated to be performed efficiently using MPC. Instead, methods derived from finding outliers in regions of interest (ROIs) are popular in radiation detection contexts and may provide a reasonable compromise between simplicity and sensitivity. Those methods still exploit the multivariate nature of spectral time series, but simultaneously reduce the parameter space to a size that is more manageable when implemented as a garbled circuit. This section discusses an ROI-based technique that was converted to a garbled circuit and applied to the MUSE dataset, creating a more robust anomaly detection algorithm while retaining the privacy-preserving elements of the interaction.

ROI methods, also referred to as energy windowing methods [113, 114], derive from early work captured in patent applications for technologies distinguishing naturally occurring radioactive material from artificially produced sources [115]. The same principles are used here to differentiate radioactive anomalies from the "natural" background. Since there is a fixed probability that any given isotope emits a gamma ray of a specific energy when it de-excites following radioactive decay, the spectral shape measured for any particular arrangement of long-lived[8] radioactive isotopes should remain generally constant (apart from natural statistical fluctuations). For similar spectral shapes, the ratio between any two channels should remain approximately the same—and so should the ratio between the total number of gamma rays measured in any two spectral ranges. In the event that an anomaly occurs, the observed spectral ratios will likely differ from their projected proportionality, and an alarm may be triggered.

Binning the data over a region in this way provides two advantages. First and foremost, the

[8]Long-lived isotopes would be any with lifetimes many times longer than the period between observations.

aggregation process may substantially reduce the number of regions that must be compared, and these regions may be chosen to focus on regions that are particularly useful in distinguishing certain anomalies from background (e.g. unplanned movements of SNM). This eliminates the combinatorically large quantity of spectral channels that would otherwise need to be compared when analyzing complete spectra for anomalies, significantly shrinking the resource load of further calculations. Moreover, the binning process also acts to smooth the data, mitigating the effect of statistical fluctuations that may arise in any particular spectral channel. These qualities make ROI-based anomaly detection techniques attractive candidates for use as garbled circuits when information is known about the characteristics of anomalies, and so the second trial performed in this study leveraged the concept of spectral comparison ratios (SCRs) developed over the past two decades [116–119].

4.5.1 The SCR Algorithm

As suggested by the name, an SCR method compares ratios of spectral counts in a set of regions. The algorithm analyzes these ratios to find cases where the counts in some combination of the regions deviate significantly from an expected value. This section explains how these ratios are computed and the metric used to evaluate them for anomalous behavior. Like in the previous trial for the gross counting technique, the algorithm computing the ratio and evaluating the metric is programmed using Obliv-C, with the resulting protocol description being compiled as a garbled circuit. The algorithmic process (and larger MPC protocol) is again represented as a diagram in Algorithm 5. The circuit generator continues to provide the spectral data (the MUSE radiation time series), while the circuit evaluator continues to provide the algorithmic parameters. In this trial, those algorithmic parameters now include the choice of regions.

The algorithm primarily relies on ratios of spectral counts determined directly from a measured spectrum of gamma-ray counts $\boldsymbol{x}_t$. Consider that spectrum partitioned into n regions collected at time t,

$$\boldsymbol{r}_t = \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix}_t . \tag{4.19}$$

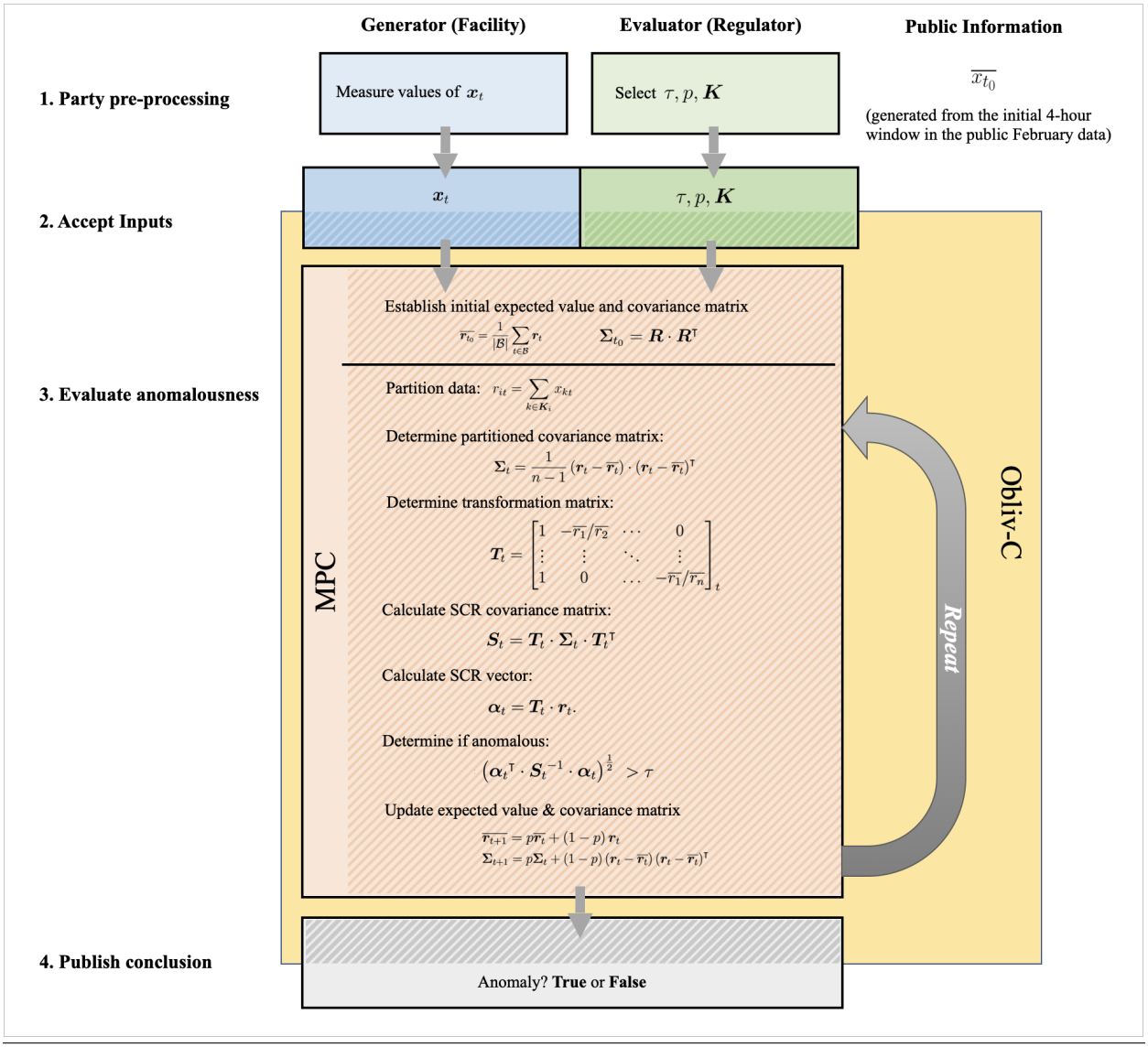
The boundaries for these regions are chosen in advance by the circuit evaluator.[9] Each value r_{it} denotes the number of counts in region i at time t, defined by

$$r_{it} = \sum_{k \in \boldsymbol{K}_i} x_{kt} , \tag{4.20}$$

where $\boldsymbol{K}_i$ denotes the set of channels comprising region i and x_{kt} counts the number of gamma rays measured in channel k.

[9]The choice of region boundaries is not important in understanding the anomaly detection algorithm's execution procedure, and so is not covered here. Boundary selection is discussed in depth in Section 4.5.2 when presenting implementation specifics used for this anomaly detection trial.

Algorithm 5 The procedural diagram for the SCR-based anomaly detection trial. The inputs on the part of the circuit evaluator now include the channels in each region $\boldsymbol{K}$.



The specific SCR method used in this work is derived from mathematical foundations proposed by Trost and Iwatschenko [115] who define a parameter $\alpha_{i,j}$ for comparing spectral counts across two regions, denoted i and j:

$$\alpha_{i,j} = r_i - \frac{\overline{r_i}}{\overline{r_j}} r_j \ . \tag{4.21}$$

Scalars $\overline{r_i}$ and $\overline{r_j}$ represent the expected values for counts in regions i and j, $\mathrm{E}[r_i]$ and $\mathrm{E}[r_j]$, respectively. This quantity represents a residual between the present observation of gamma-ray counts in a region and the expected number of counts in that region, the latter estimated by using the measured counts in another region together with the historical relationship between the two [117].

For a partitioned spectrum with n regions, there exist $n-1$ linearly independent such residuals between the regions. At time t, one possible vector of these $n-1$ residuals is

$$\boldsymbol{\alpha}_t = \begin{bmatrix} \alpha_{1,1} \\ \vdots \\ \alpha_{1,n-1} \end{bmatrix}_t \ . \tag{4.22}$$

Given the vector of gamma ray counts in each region $\boldsymbol{r}_t$, the residual vector for time t is calculated as

$$\boldsymbol{\alpha}_t = \boldsymbol{T}_t \cdot \boldsymbol{r}_t. \tag{4.23}$$

where $\boldsymbol{T}$ is a transformation matrix defined to generate a set of SCR-based residuals from ROI counts:

$$\boldsymbol{T}_t = \begin{bmatrix} 1 & -\overline{r_1}/\overline{r_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \ldots & -\overline{r_1}/\overline{r_n} \end{bmatrix}_t \ . \tag{4.24}$$

By constructing this transformation matrix, the algorithm may efficiently convert between spectra partitioned into ROIs and SCR-based residual vectors.

With the notion of SCRs defined, it remains for the SCR anomaly detection algorithm to be implemented as a continuously updating prediction model. To do that, each new measured gamma ray spectrum must be partitioned into regions, converted into a vector of SCR-based residuals, and then evaluated for anomalous behavior. Similar to the threshold values described in Sections 4.3 and 4.4, a spectrum could be identified as anomalous if, under some metric, it exceeded a threshold τ. Expressing the metric as some arbitrary distance function, an anomalous spectrum with SCR vector $\boldsymbol{\alpha}_t$ would satisfy the relationship

$$\mathrm{dist}\left(\boldsymbol{\alpha}_t\right) > \tau \ . \tag{4.25}$$

Implementing Equation 4.25 requires a defined multivariate distance metric and the associated inputs. To avoid high-count regions from dominating, a metric accounting for correlations is preferable. In this case, a distance function related to the Mahalanobis distance introduced in in Equation 4.6 serves as a reasonable metric [116, 118]:

$$\mathrm{dist}\left(\boldsymbol{\alpha}_t\right) = \left(\boldsymbol{\alpha}_t{}^{\mathsf{T}} \cdot \boldsymbol{S}_t{}^{-1} \cdot \boldsymbol{\alpha}_t\right)^{\frac{1}{2}} \ . \tag{4.26}$$

In this equation, $\boldsymbol{S}_t$ defines the covariance matrix of SCR residuals in the BSP. This covariance matrix is derived from the covariance matrix of partitioned BSP counts $\boldsymbol{\Sigma}_t$, and that may be calculated directly from the spectral counts measured in each region. Using the standard (unbiased) formula for building a covariance matrix from a dataset, this is

$$\boldsymbol{\Sigma}_t = \frac{1}{n-1} \left(\boldsymbol{r}_t - \overline{\boldsymbol{r}_t}\right) \cdot \left(\boldsymbol{r}_t - \overline{\boldsymbol{r}_t}\right)^{\mathsf{T}} \ . \tag{4.27}$$

In this equation, $\overline{\boldsymbol{r}_t}$ is a vector representing the n values of $\overline{r_i}$, the expected number of counts in each region. Once this covariance matrix has been calculated directly from the data, the SCR covariance matrix $\boldsymbol{S}_t$ can be found using the transformation matrix $\boldsymbol{T}_t$ defined in Equation 4.24:

$$\boldsymbol{S}_t = \boldsymbol{T}_t \cdot \boldsymbol{\Sigma}_t \cdot \boldsymbol{T}_t^{\mathsf{T}} \ . \tag{4.28}$$

The exact, ordered procedure for determining $\mathrm{dist}\left(\boldsymbol{\alpha}_t\right)$ is shown in Algorithm 5.

In order for this residual comparison to be implemented as a continuously updating prediction model, the relationship between the model and the BSP must also be defined. This is critical since estimates of $\overline{r_{it}}$ are necessary to build and update matrix $\boldsymbol{T}_t$, and the continually revised BSP is the source underlying the covariance matrix $\boldsymbol{\Sigma}_t$. Recognizing that the initial gross counting method suffered from an inability to adapt to low frequency changes in the MUSE data, this second analysis used EWMAs from the outset to update information gleaned from the BSP. This procedure, analogous to the one described in Section 4.4.4 and Equation 4.18, was previously shown to work well with the SCR anomaly detection technique [118], and so was a natural choice. As in Section 4.4.4, the EWMA allows the anomaly detection method to incorporate new data into the BSP while simultaneously preserving long-term history of the dataset's background and ensuring that recently acquired data are appropriately weighted in the evaluation.

By definition, only non-anomalous data may ever be included in the BSP. As such, it is only for spectra where $\boldsymbol{\alpha}_t$ is determined to be non-anomalous by Equation 4.25 that the BSP averages are adjusted. First, for the expected value of counts in each region i, the EWMA average is calculated as

$$\overline{\boldsymbol{r}_{t+1}} = p\overline{\boldsymbol{r}_t} + (1-p)\,\boldsymbol{r}_t \ . \tag{4.29}$$

Similarly, the covariance matrix is estimated using the relationship

$$\boldsymbol{\Sigma}_{t+1} = p\boldsymbol{\Sigma}_t + (1-p)\left(\boldsymbol{r}_t - \overline{\boldsymbol{r}_t}\right)\left(\boldsymbol{r}_t - \overline{\boldsymbol{r}_t}\right)^{\mathsf{T}} \ . \tag{4.30}$$

The equations describing these two iteratively updated inputs share parameter p, the same *persistence* parameter that was defined in Section 4.4.4. When calculating the new EWMA, the persistence ($0 \leq p < 1$) is used to weight the previous background average against the present measurement's contribution as it is incorporated into the average. Ultimately, the persistence determines how substantially previously collected measurements influence the current evaluation over time.[10] If, on the other hand, $\mathrm{dist}\left(\boldsymbol{\alpha}_t\right) > \tau$ and the spectrum collected at time t is classified as anomalous, the BSP is left unchanged and the previous averages are carried forward: $\overline{\boldsymbol{r}}_{t+1} = \overline{\boldsymbol{r}}_t$ and $\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t$.

[10] Note that the persistence is the fractional complement of the weighting factor used by Pfund et al. [118]; that weighting factor is equivalent to $1-p$.

Now, since both the expected value and ROI covariance matrix are recursively defined, an initial value must be determined using a predefined initial BSP for time $t = t_0$. That BSP is used to compute the initial vector of expected values for region counts $\overline{\boldsymbol{r}_{t_0}}$ using the traditional vector mean of the values in each partitioned spectrum, and likewise, to perform the standard calculation of the covariance matrix:

$$\boldsymbol{\Sigma}_{t_0} = \boldsymbol{R} \cdot \boldsymbol{R}^{\intercal}, \quad \boldsymbol{R} = \begin{bmatrix} \vdots \\ \boldsymbol{r}_t - \overline{\boldsymbol{r}_t} \\ \vdots \end{bmatrix}, \quad t \in \mathcal{B}. \tag{4.31}$$

Updating the expected values and covariance matrix using this technique is also included in the diagram of Algorithm 5, depicting how the update process fits into the evaluation MPC protocol.

4.5.2 Identifying MUSE Anomalies Using Spectral Comparison Ratios

Following the same general procedure as in the first trial, this ROI anomaly detection algorithm was applied to the MUSE dataset using a garbled circuit to preserve input privacy. Again, Obliv-C was used to compile garbled circuits of the algorithm outlined in Section 4.5.1 from a protocol description written in C. Using the garbled circuit executable, two parties were simulated engaging in a joint privacy-preserving computation in the roles of a nuclear facility operator and a safeguards administrator.

Although the private inputs for the nuclear facility still consisted of data collected by MUSE detector 4, this second trial analyzed data for just the month of March. The nuclear regulator's input increased, however, now being composed of three items: the anomaly detection threshold τ, desired persistence p, and the boundaries of each region used to partition the spectra. These three inputs had to be determined in advance by the party in the role of the safeguards administrator, and so all of the MUSE data from the month of February was made public to facilitate the process. In a real scenario, this would represent the case where the regulator is given access to a facility and some initial period of operation, perhaps including some set of calibration or testing events. The facility would not learn the parameters chosen by the regulator to ultimately derive its conclusions, it would know only the data that the regulator had available in making its choices. Likewise, the regulator would not gain access to facility data collected after this initial window.

First, the party simulating the safeguards administrator had to select an appropriate set of regions for the algorithm to use when partitioning each measured spectrum. As this selection process was not included as part of the garbled circuit, it was not described in Section 4.5.1; however, thoughtful selection is critical for maximizing the likelihood of detecting anomalies of interest. While the region selection process is specifically *not* designed to identify particular isotopes in the dataset [118], regions may be selected to enhance the detection of anomalies that share common characteristics with potential target isotopes.

The selection procedure used here followed the outline set forth by Anderson et al. [117], determining the optimal region boundaries to provide the maximum discrimination ability between anomalous and non-anomalous spectra. To hone the algorithm's ability to distinguish events in this way, the technique relies on a designated "threat" spectrum intended to be representative of likely

anomalies. In this study, isotopes of interest for composing a threat spectrum might include neptunium, protactinium, or plutonium, three SNM material sources like those being transferred in the manufactured anomalous events within the MUSE dataset. Many of the material and experiment characteristics pertinent to the MUSE dataset (e.g. relative abundances, transportation shielding, standoff distance, etc.) were not known, and so generating a reference threat spectrum for a relevant transfer was impractical. Instead, since the data from the month of February was considered public, this second anomaly detection trial made use of data and results from the previous trial. In place of an experimentally measured or artificially simulated threat spectrum, the most anomalous spectrum from the first trial (corresponding to the known neptunium and protactinium transfer event) was used to select optimal regions.

As a note, while this spectrum was included in the now-public February data, had that data not been public, the same information could have been extracted using additional steps in the privacy-preserving algorithm. For instance, a flag could be included in a preliminary privacy-preserving scan to track and mark the most anomalous events encountered during the program over the course of one month. That event's data could then be used in a subsequent privacy-preserving computation to optimize region boundaries. Once determined, these privately generated region boundaries could be fed back into the on-line anomaly detection algorithm. This course of action was not taken here, as the optimization process was found to already be quite computationally intensive and the loss of privacy for the February data did not sacrifice the integrity of any of the March data.

With a threat spectrum defined, the region optimization procedure searches to find the set of regions that distinguishes the threat spectrum from background events with the minimum number of detectable counts. This minimum number may be taken as inversely proportional to a figure of merit (FOM) that quantifies how any configuration of regions impacts the algorithm's capacity to identify the threat. By maximizing the FOM for the chosen regions, the selection procedure is exploiting the fact that gamma rays measured in certain areas of the spectrum contribute more strongly towards a determination of abnormality than others. Given a threat spectrum $\boldsymbol{x}_{\text{threat}}$, the FOM is determined to be

$$\text{FOM} = \boldsymbol{\alpha}_{\text{threat}}{}^{\intercal} \cdot \boldsymbol{S}_t{}^{-1} \cdot \boldsymbol{\alpha}_{\text{threat}} \, . \tag{4.32}$$

Here, $\boldsymbol{S}_t$ is the SCR covariance matrix of the BSP as calculated in Equation 4.28 and $\boldsymbol{\alpha}_{\text{threat}}$ is the SCR vector of the threat spectrum. Vector $\boldsymbol{\alpha}_{\text{threat}}$ is generated by first partitioning $\boldsymbol{x}_{\text{threat}}$ into test regions $\boldsymbol{r}_{\text{threat}}$, before applying the transformation matrix $\boldsymbol{T}_t$ defined for the BSP at time t. This FOM should be maximized for the complete set of combinations of input bins and bins may overlap.[11] For considering multiple potential threat spectra, the notion may be generalized [116, 118].

Applying Equation 4.32 to various regions in order to optimize the FOM also requires an initial BSP so that the covariance matrix $\boldsymbol{S}_t$ may be determined. Just like in the first trial, this BSP was selected to be the first 14,399 spectra from the February data. Using this BSP, along with the most anomalous February spectrum, three (overlapping) regions were selected as ideal. It is worth noting that to reduce the computational burden of both the optimization process and the privacy-preserving computation, the 500-channel spectrum was down-binned into 25 bins of 20 channels each, a level

[11] It is also evident, though not necessarily immediately obvious, that these bins *must* be linearly independent. If not, covariance matrix $\boldsymbol{\Sigma}_{\mathcal{B}}$ (and therefore, by extension, $\boldsymbol{S}_{\mathcal{B}}$) will be a singular matrix and non-invertible.

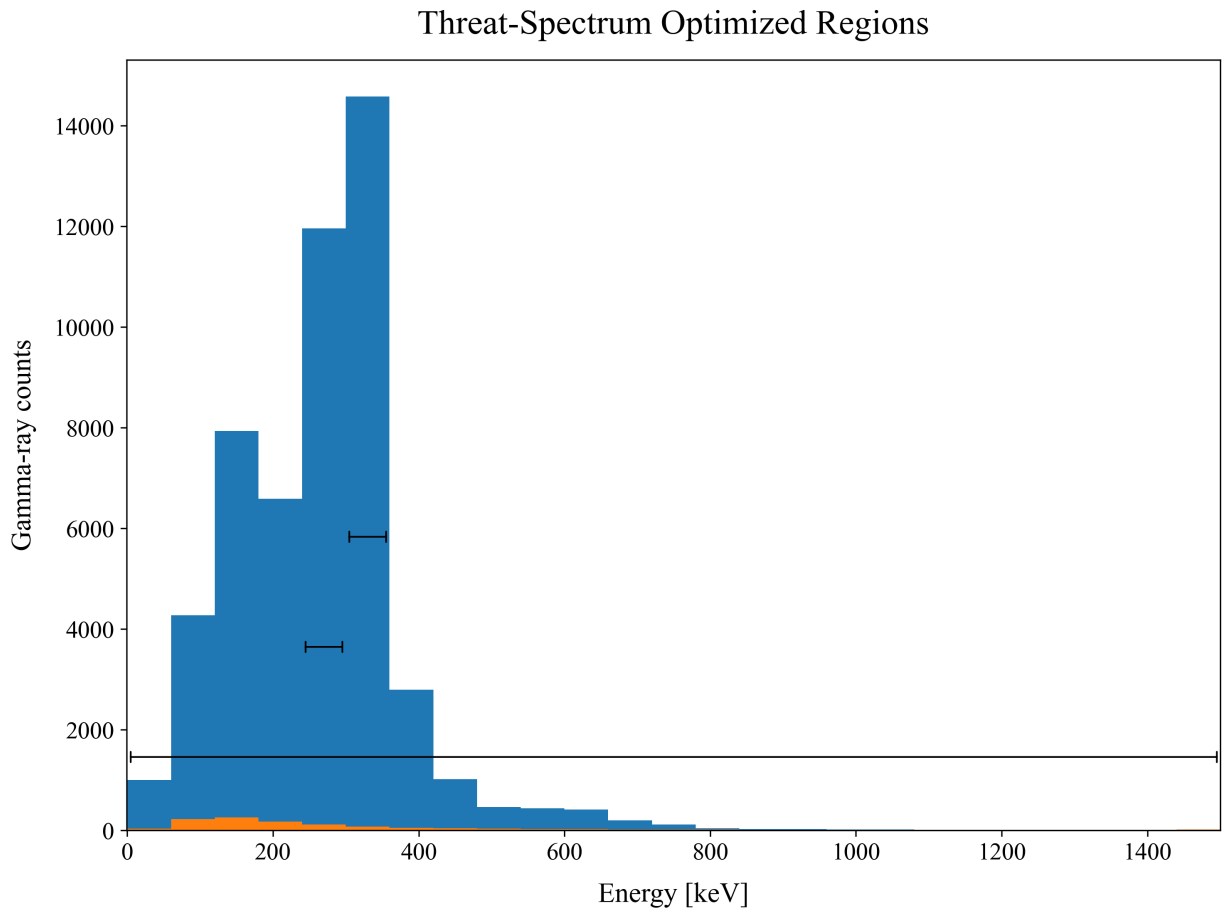


Figure 4.4: A spectrum from the MUSE dataset representing a transfer of neptunium and protactinium between REDC and HFIR. This was used as the reference threat spectrum for optimizing the ROIs; the three regions determined to optimally discriminate anomalies from non-events are overlaid on the spectrum as solid horizontal lines (with arbitrary positions on the vertical axis). The regions span the regions of 0 keV–1500 keV, 240 keV–300 keV, and 300 keV–360 keV. The average initial background measurement is overlaid in orange at the bottom.

of granularity that retained the most notable spectral features. The chosen "threat spectrum" and the selected regions are shown in Figure 4.4, and it is evident that the regions follow some of the features in the dataset. More regions could potentially improve the algorithm's ability to discriminate between spectra; however, this study capped the algorithm to only three regions to prevent excessively long execution times. When considering the implementation of the SCR-based anomaly detection algorithm as a garbled circuit, computational limits restrict the number of regions that can be feasibly analyzed. Furthermore, the optimization process becomes computationally infeasible with large spectra and many potential groupings of regions, although optimization tools and

algorithms can improve the efficiency of this process.[12] For the down-binned spectrum and three possible regions it was possible to exhaustively search all 5,666,050 potential region configurations to find the set that maximized the FOM.

Once the regions had been identified, it remained for the inspecting party to determine the other two inputs: threshold τ and persistence p. The threshold value was set by hypothesizing that anomalies of interest (transfer events) would closely resemble the spectrum of the known transfer event. Events matching that description would be those with similarly high values calculated with the SCR distance metric $\mathrm{dist}\,(\boldsymbol{\alpha}_t)$. The objective became distinguishing these transfer events from other unusual—but not alarming—features in the collection of spectra. To achieve this, τ was set comfortably larger than the maximum distance value from the February data that could not be verified on the ground truth logs as a transfer event, yet not so large as to reduce the sensitivity of the algorithm.

Following these criteria, the threshold was set at $\tau = 100$. The distances calculated for the true anomalous event[13] were well above this threshold, with most of those values being greater than one thousand (and in some cases, larger than one million). At the same time, for $\tau = 100$, the largest non-anomalous distance in the February dataset for detector 4 had a computed distance of only $\mathrm{dist}\,(\boldsymbol{\alpha}_t) \approx 55$—still significantly less than the threshold.

Finally, the algorithm was applied to the MUSE data from the month of March. Since the February data were deemed public, they were used to initiate the anomaly detection protocol. The protocol was run through the month of February in order to establish a history through the EWMAs before proceeding forward to analyze the private March data for anomalies.

4.5.3 Results

Using Obliv-C, the garbled circuit implementation of the SCR anomaly detection algorithm was successfully implemented. It was able to compute and identify anomalous events from the March MUSE data provided by the simulated nuclear facility, using the private parameters supplied by the simulated regulator.

The privacy-preserving anomaly detection technique based on SCRs was highly effective, finding a total of seven anomalies during the month of March. The first three of these anomalies occurred slightly after 10:56 on March 6, and the next four anomalies occurred at 10:36 on March 13. These two events corresponded to the only material transfers of SNM recorded in the ground truth logs. In this case, the two events consisted of irradiated neptunium and plutonium targets moving past MUSE detector 4 en route to HFIR from the REDC facility. These two anomalous event windows were both identified with high confidence, having maximum distance metric values substantially greater than the $\tau = 100$ threshold; for each event, at least one spectrum was calculated to have a distance metric above one thousand.

[12] Powell's Method has been suggested as an effective optimization method for ROI selection [117].

[13] Specifically, spectra collected within about a one-minute interval surrounding the February 27 transfer event.

4.6 Summarizing Remarks

Both of these trials showed how garbled circuits could be used to apply an anomaly detection algorithm on the MUSE dataset without revealing information about the input spectra. The first presented a relatively simple approach using the gross number of gamma rays measured in each spectrum to find anomalous material transfer events, while the second offered a significantly more powerful technique using ROIs to analyze a similar set of spectra. In each of the trials, the algorithm successfully identified spectra corresponding to the material transfer events serving as anomalies in the data. When the models were refined, the algorithms did not just identify the anomalous events, but were also able to minimize the number of false positives that were incorrectly identified.

Neither of the two trials featured anomaly detection systems representing the most robust spectral analysis that could be performed on the data, though enhancements could be made within the garbled circuit framework without much difficulty. For example, it is possible to filter out nuisance spectra from the input dataset when performing anomaly detection using the SCR method [116, 118]. These nuisances could include artificially produced radioactive products that contaminate a set of test spectra and make identification of anomalies based on other isotope signatures difficult. If radioactive isotopes like argon-41—emitted intermittently from the HFIR stack [101]—and its progeny were believed to obscure the signal from the motion of SNM, those isotopes could be characterized as nuisance candidates and filtered out by the anomaly detection algorithm. Although this study did not incorporate any nuisance filtering in its finalized algorithm, this functionality could be added by adjusting $\boldsymbol{\alpha}_t$ in Equation 4.26.

Ultimately, with near universal identification of anomalies and low false alarm rates, both trials proved successful in applying MPC to perform privacy-preserving calculations on nuclear safeguards data. Without revealing any information about the underlying gamma-ray spectra data to the regulator, and without revealing the chosen regions of interest, threshold value, or any historical dependence of background data to the nuclear facility, the algorithms were able to confidently identify material transfers from other potential data irregularities as notable anomalies. The success of these exercises demonstrates how MPC might be applied to nuclear safeguards challenges more generally, and suggests avenues for further exploration, study, and advancement.

Chapter 5

CypherCircuit

Having demonstrated that nuclear safeguards challenges *can* be practically overcome through the use of multiparty computation (MPC) techniques, it is appropriate to consider *how* privacy-preserving computation might be actually adopted by the International Atomic Energy Agency (IAEA) and the international safeguards community. Like many stakeholders in nuclear operations worldwide, the IAEA chooses to upgrade its technological systems conservatively, ensuring that risks are not introduced. For safeguards, rigorously evaluated technologies are trusted to inform accurate safeguards assessments, even at the expense of efficiency or convenience. At the same time, nuclear facilities and the States that operate them are likewise risk averse. Maintaining the privacy of proprietary data and reducing liability is perceived to be worth more than the potential benefits of implementing a state-of-the-art system. Adoption of new technologies may also be stymied by high volumes of administrative overhead required under what are often comparatively intensive regulatory environments, where new technologies must be proven safe and effective. Then, taking this perspective, it becomes critical that both the IAEA and the States participating in safeguards possess high confidence in the security of any proposed MPC solution.

5.1 A New Garbled Circuit Framework

Numerous computational frameworks already exist to perform MPC, as discussed in Section 3.5. Each is designed with a specific purpose in mind, and the ideal framework to choose is almost always dependent on the task at hand. Many are cryptographic research tools, designed to either demonstrate a new technique or compare the performance of a selection of MPC protocols. More recently, some of the frameworks have been developed with the intention that they be usable by general practitioners—non-experts without extensive, formal cryptographic training. Obliv-C, the software framework used to perform the analysis demonstrations in Chapter 4, is one such tool [99]. Despite this emerging trend emphasizing usability, none of the codes are intended to elucidate the underlying details of the MPC processes taking place. All of the implementations, even those described comprehensively in the academic literature, are rather opaque in their operation. This is especially true for users without cryptography experience.

For many applications, this arrangement is sufficient. There are many users who would likely be

satisfied with the knowledge that a MPC tool uses techniques that have been published and proven in the academic literature and verified by cryptographic experts. However, in other situations there may be users who do not completely understand the underlying MPC concepts—or even just users who do not understand exactly how the cryptographic protocol is translated into a software library—and who would therefore be reluctant to trust a computational solution based on algorithms that are difficult to grasp or that are not clearly demonstrated by a tool executing with complete transparency.

International regulators and States participating in safeguards agreements might both fall into this second classification of users. From the perspective of the IAEA, the agency must be assured that any use of MPC techniques does not expose vulnerabilities in either new or existing safeguards infrastructure. At the same time, limited resources may prevent the agency from devoting substantial effort toward developing the in-house expertise for understanding MPC protocols, verifying the software tools that execute those protocols, and convincing States to embrace these algorithms. Considering the opposing perspective of a State participating in safeguards, the State may be wary that nominally privacy-preserving protocols might actually be vulnerable to manipulation that could leak their private information. In order for either of these entities (or any skeptical party) to build trust in MPC protocols, it is essential that the algorithms be presented in a manner that is as clear and straightforward as possible, providing evidence along the way of the algorithm's security. Demonstrations like those in Chapter 4 only demonstrate that privacy-preserving computation *could* be used in a nuclear safeguards context, but alone may do little to reassure a skeptical party that they *should* consider adoption of MPC. The *CypherCircuit* software library was created to satisfy this latter objective.

5.2 The *CypherCircuit* Package

The *CypherCircuit* Python package provides users with an intuitive tool for understanding garbled circuits. Its purpose is to make the underlying garbled circuit protocol clear and transparent to users, allowing them to engage more intimately with the protocol they are executing. The intention of developing a software tool in this way is to facilitate trust in the algorithms by the participants. Parties who construct, encrypt, and evaluate garbled circuits should be able to investigate each step of the process to understand exactly how their inputs are being manipulated—and therefore, how their privacy is being preserved and that their results are trustworthy. Still, functionality cannot be ignored. While the *CypherCircuit* package is not designed to match the speed of modern research and production codes, it should be able to handle moderately sized calculations.

5.2.1 Design Principles

To successfully develop a software framework that met our objectives, the *CypherCircuit* project has attempted to adhere to a set of three guiding principles, unified by a general commitment to intuitive design. These three principles are accessibility, perspicuity, and extensibility.

Accessibility Considering that the *CypherCircuit* package is targeted at technical users without extensive cryptography experience, accessibility is a critical characteristic of the framework. It can be expected that users are familiar with a general understanding of garbled circuits, since the operations are readily presented at varying technical levels in the academic literature, as well as in less formal contexts (e.g. freely published online notes, textbooks, and even technical blogs). With this consideration, it becomes important that the *CypherCircuit* package be constructed in a way that logically follows from those explanations. Using *CypherCircuit*, a user should be able to grasp the premise of garbled circuits, then translate those ideas directly into software that executes two-party computation (2PC). *CypherCircuit* is designed to facilitate this process of circuit prototyping; it offers intuitive circuit constructions and operations that cleanly map to common descriptions of garbled circuit protocols. Moreover, the package attempts to use intuitive data structures that blend a low-level understanding of garbled circuit cryptography and circuit design with high-level mathematical operations, attempting to seamlessly integrate both sides of the MPC to facilitate an understanding of how simple privacy-preserving exercises may be extended to more sophisticated calculations.

Now, although being intuitive from a conceptual standpoint is important, it is not the only aspect that should be taken into account when focusing on accessibility. The larger context of usability must also be considered, and users should find the development environment manageable and straightforward. Towards this end, the *CypherCircuit* project emphasizes attention to detail. First, *CypherCircuit* adheres strictly to applicable software development standards (e.g. Python PEP guidance) to ensure that the programming interface is largely consistent with the greater Python scientific computing domain. For those already familiar with the Python standards, following these common patterns streamlines the development process and simplifies knowledge acquisition and code exploration. For users who are less familiar with the Python standards, the package follows structures that are common across the Python scientific computing ecosystem improving the odds that community resources may apply in interpreting code features. Second, the package is extensively documented to avoid ambiguity about the software's functionality. Thorough documentation ideally gives users a clear starting point for understanding the software—either when reading source code or when attempting to build live circuits. Third, the package is designed to anticipate developer needs and alleviate confusion. As mentioned, the package strives to offer a hybrid development approach that can construct circuits at either high or low levels. Great care was taken in the development process to prepare interfaces that allow a user to seamlessly work in either context, with the software's back end ensuring consistency between both approaches. When taken all together, these components are critical for enabling users to use the package confidently: able to easily work with the package while being reassured that they are not leaving themselves exposed to security vulnerabilities through accidental misuse.

Perspicuity The *CypherCircuit* package also emphasizes that clarity and transparency be afforded to users at runtime. In the safeguards scenario, the intended users of the package (the IAEA and a State under safeguards) ought to be assumed skeptical or wary, and so offering clear and unencumbered access to inspect the software's execution is highly worthwhile. *CypherCircuit* en-

deavors to offer this transparency by facilitating user interaction and exploration of the garbled circuits throughout the entire multi-step garbled circuit process. In this way, *CypherCircuit* leverages Python's permissive access structures to enable users to inspect circuits at various stages of the garbled circuit protocol—whether during construction, encryption, exchange, or evaluation. While this aspect is likely of minimal importance to large scale operations, low-level access to the package's machinations permits a skeptical party to match *CypherCircuit*'s execution to the expected garbled circuit protocols in smaller test cases. This degree of accessibility is extremely valuable given the maturity of interactive Python tools—the Python interpreter, IPython, Jupyter notebooks, and JupyterLab, to name a few. Further description of the measures taken by *CypherCircuit* to enhance transparency are described in Section 5.2.2.

The *CypherCircuit* package's commitment to transparent code and execution procedures is not limited to implementations and applications of the package. To ensure the package's consistent integrity, the *CypherCircuit* project is also comprehensively tested, tracked via version control, and maintained using continuous integration. These practices enable users to have high confidence in the code's ability to perform computations securely, as well as offer a detailed window into the software's execution and evolution.

Extensibility The third, final design principle behind *CypherCircuit* is extensibility. To be useful for general audiences, as well as support an intuitive package design, *CypherCircuit* must be easily extended to handle a broad range of (potentially unforeseen) user demands. As such, most elements of the *CypherCircuit* package are intended to serve as foundations—but not the limits—of the tool's capabilities. Examples of this philosophy are readily found in the package's components. First, the package offers a basic set of generic garbled circuit tools, which can then be optimized or consolidated using logical extensions and adaptations of the existing objects. For example, the package relies on `Wire` and `Gate` objects for building simple, conventional circuits, but these elementary structures can be concealed by using the higher-level and more intuitive language-like `CCType` system (covered in Section 5.2.2) or optimized using "smart" features like the `Hi`, `Lo`, and `FreeXorWire` objects (described in detail in Section 5.2.3). Whenever possible, *CypherCircuit* seeks to define interfaces that enable the package's built-in objects to be easily extended to tackle new challenges.

5.2.2 Package Structure

In pursuit of an intuitive design, the most basic elements of the *CypherCircuit* package have been developed to be used as if one were building a real, physical digital circuit. A similar approach has been used in other MPC libraries; for instance, some common features are shared with the library presented by Huang et al. [68]. *CypherCircuit* provides a toolset capable of executing the four stages of the two-party protocol outlined in Section 3.2.1: constructing, garbling, exchanging, and evaluating the circuit. To begin, the two parties build and agree to use a circuit structure that evaluates the function to be jointly computed. They then proceed to engage in an interactive process guided by a Python context manager that handles the second, third, and fourth stages of the protocol, as well as the associated inter-party communication. The package handles all operations,

storage, encryption, and communication—the user simply triggers those actions using the built-in functionality of the circuit. As Python objects, almost all elements can be interrogated by a user to expose the underlying data (of course, without compromising the security of the algorithm).

Constructing the Circuit

To construct garbled circuits with *CypherCircuit*, a user begins by instantiating a `CircuitBoard` object. This `CircuitBoard` serves as the underlying command structure for each circuit, storing circuit level attributes like the security parameter, as well as providing methods for manipulating, garbling, and sharing the circuit. The user then adds wires and gates to this `CircuitBoard` object, implementing the desired functionality. Like a physical circuit, these wires and gates form a directed acyclic graph (DAG) with wires as edges and gates as nodes. `Wire` objects are relatively compact data structures, each able store one of two potential values representing the Boolean logic value carried by the wire, as well as the associated labels. Gate objects operate on their input wires, propagating the wire values through the circuit. A comprehensive set of built-in gate types are provided by the package (`NOT`, `AND`, `OR`, `NAND`, `NOR`, `XOR`, and `XNOR`), all implementing a common gate interface. Each gate is instantiated by passing the gate's input wires as arguments, and, if not otherwise specified, automatically produces a corresponding output wire. Figure 5.1 displays the process that a user would take to programmatically build a comparator circuit using *CypherCircuit*.

All gates enforce a logic operation defined by a truth table, an object that is common to all gates of a given type. While most common gates are predefined, truth table objects may be constructed dynamically by defining gates that perform arbitrary operations. This functionality is one example of the extensibility principle being applied in practice. Gates and wires exist with knowledge of adjacent circuit components, such that when wire values are defined on all of a gate's input wires, the gate sets the value of its output wire accordingly. In fact, these types of gate updates are triggered automatically whenever wire values are changed, enabling full circuits to be immediately responsive to adjustments on input wire values and propagating those changes through to the circuit output wires. Users may leverage this property to ensure that a circuit is computing a function correctly before proceeding through the more computationally expensive encryption stages of the protocol.

Although storing each and every wire and gate object is not the most memory-efficient construction, providing a user with the flexibility to implement functions by building up from the circuit level enables *CypherCircuit* to deliver on its guiding principles. First, by retaining the obvious connection to a circuit structure, non-expert users familiar with the concept of garbled circuits can quickly begin exploring the protocol's operation (accessibility). Second, users may easily investigate and inspect the state of wires and gates during each step of the garbling process (perspicuity). Third and finally, allowing users to build arbitrary circuits guarantees that any computable function can be implemented by the package, and users may optimize those circuits freely (extensibility).

In traditional garbled circuit calculations, both parties know the function to be computed and therefore may possess knowledge of the circuit structure. As such, *CypherCircuit* allows parties to share common circuits in two manners. The first option is for a user to define a circuit as an element of a standalone Python module. This circuit, ideally derived from the built-in `CircuitBoard` class, may be subsequently imported into the execution environment of each party. A second option per-

```
# Load CypherCircuit components
from cyphercircuit import CircuitBoard
from cyphercircuit.wires import Wire
from cyphercircuit.gates import And, Or

# Create the comparator circuit
comparator = CircuitBoard()

# Add comparator input wires
wire1, wire2 = Wire(comparator), Wire(comparator)

# Feed the wires into the circuit's gates
gate1 = Not(wire1)
wire3 = gate1.output_wire
gate2 = Not(wire2)
wire4 = gate2.output_wire
gate3 = And(wire2, wire3)
wire5 = gate3.output_wire
gate4 = And(wire1, wire4)
wire6 = gate4.output_wire
gate5 = Nor(wire5, wire6)
wire7 = gate5.output_wire
```

Figure 5.1: The steps taken to generate a comparator circuit like the one diagrammed in Figure 3.1 using the *CypherCircuit* package.

mits a circuit to be generated entirely by one party (the circuit generator) and then transmitted via an online network connection to the other (the circuit evaluator) using the protocol context managers. For transmission, the circuit is first converted into a serializable JavaScript Object Notation (JSON) representation of the DAG composed of circuit elements—wires and gates—intuitively named a circuit *diagram*. This diagram is communicated to the evaluator, who is then able to reconstruct an identical circuit using functionality provided by the package.

Garbling the Circuit

With a circuit constructed, a user possesses the ability to control and explore each step of the garbling process. Users may exert control of the circuits and garbling protocols manually by using predefined methods built in to each `CircuitBoard` object, although many users may prefer to leave these execution details to one of the package's protocol context managers. The *CypherCircuit* package provides two of these context managers, a `Generator` and an `Evaluator`, to guide parties through the interactive garbling protocol. The following description outlines the individual

```
>>> circuit = CircuitBoard()
>>> X, Y = Wire(), Wire()
>>> and_gate = And(X,Y)
>>> and_gate.label()
>>> and_gate.labeltable
 W0001    W0002   |   W0003
--------------------------
aeef0d...  10e1a1...  |  2a9d8f...
aeef0d...  0844fa...  |  2a9d8f...
b64a56...  10e1a1...  |  2a9d8f...
b64a56...  0844fa...  |  3238d4...
```

Figure 5.2: A labeled truth table belonging to a *CypherCircuit* `AND` gate. Wire labels are (abbreviated) hexadecimal strings representing 128-bit binary integers.

steps that must be taken by a manual circuit garbler, although these (and all subsequent) steps are conveniently bundled together for users taking advantage of the context managers.

Following the garbled circuit procedure described in Section 3.2.1, each `Wire` object in the circuit may be assigned two random labels corresponding to its two potential values. The labels that are generated in this process follow the global offset method described in Section 3.2.2, where labels are randomly selected for one wire label and then that label is used as a one-time-pad to encrypt the randomly generated global key offset. This practice allows *CypherCircuit* to leverage the benefits of the *FreeXOR* technique, an option described in Section 5.2.3. In *CypherCircuit*, all randomness is generated using cryptographically secure random number generators from the Python built-in `secrets` module. The labels for any given wire may be accessed by using attribute lookup on the corresponding `Wire` object, or a labeled truth table for a gate may be produced by the corresponding gate object. An example of a truth table labeled in this way is shown in Figure 5.2 as it would be accessed via the Python interpreter; note the similarity with the labeled truth table table depicted in Figure 3.3a.

Once labels have been assigned to all wires in a circuit, the truth tables for each gate may be encrypted. Each *CypherCircuit* gate has the ability to trigger an encryption process for the gate's (labeled) truth table following the garbled circuit protocol outlined in Section 3.2.1. For each row in the table, the algorithm symmetrically encrypts the gate's output wire labels with the corresponding input wire labels. Once each row is encrypted, the resulting ciphertext token is stored in a separate table accompanying the gate. These encrypted tables of ciphertexts resemble the table described in Figure 3.3b, with Figure 5.3 depicting one such table.

In general, *CypherCircuit* implements the non-free encryption scheme presented by Kolesnikov and Schneider [66]. As referenced in Equation 3.7, given gate g_j with input wires w_a, w_b and output

```
>>> circuit = CircuitBoard()
>>> X, Y = Wire(), Wire()
>>> and_gate = And(X,Y)
>>> and_gate.encrypt()
```

```
>>> and_gate.tokentable
ciphertexts
-----------
  640785...
  77cafd...
  0e7726...
  30bba6...
```

Figure 5.3: A encrypted truth table belonging to a *CypherCircuit* `AND` gate. Like the labels presented in Figure 5.2, tokens are (abbreviated) hexadecimal strings representing 128-bit binary integers.

wire w_c, the encryption function is defined for output wire label $\ell_c^{v_c}$ as

$$E_{\ell_a^{v_a},\ell_b^{v_b}}\left(\ell_c^{v_c}\right) = H\left(\ell_a^{v_a}||\ell_b^{v_b}||j\right) \oplus \ell_c^{v_c}\,, \tag{3.7}$$

where the $||$ operator denotes string concatenation. In the same way, output wires are encrypted as in Equation 3.8. *CypherCircuit* approximates the behavior of random oracle H using the SHA-256 hash function. This is a practical design decision, not a requirement, and the library is constructed to allow other hash functions to be used as drop-in replacements.

Finally, any gate that has been encrypted may be shuffled randomly. Since *CypherCircuit* relies on the point-and-permute method to reduce computation, shuffling is performed by permuting the circuit according to the permutation bits assigned to each wire label (as described in Section 3.2.2). Using this techiniqe, each table of ciphertexts is ordered according to the randomly generated permutation bits that match the input wire labels used as encryption keys. Permutation bits are randomly generated, and so the resulting table represents a random permutation of the ciphertexts.

Within the *CypherCircuit* framework, the three actions of labeling wires, encrypting labels, and shuffling the ciphertexts are collectively referred to as "garbling". For curious users wishing to maintain control over the garbling process, each action may be performed individually for each wire (in the case of labeling) or gate (in the case of encryption and permutation). Recognizing that many users may want to perform these actions for an entire circuit, however, methods of the `CircuitBoard` class enable these actions to be performed simultaneously for all wires and gates across the circuit. Leaving both methods available to users is designed to improve both the framework's accessibility and transparency. The code is easier to use without needing to manually perform each encryption and permutation, yet users may take advantage of the ability to exercise fine grain control to analyze or customize each circuit operation.

Exchanging the Garbled Circuit and Private Inputs

After a circuit has been garbled by the generating party, the garbled circuit protocol dictates that it must be exchanged with the evaluating party. This coordinated exchange of information is an essential aspect of *CypherCircuit*. Transmission of both the circuit garbling and encoding, along with all inter-party communication, is performed over an online network using the Transmisssion Control Protocol (TCP). All of this online execution is controlled entirely by the *CypherCircuit* context managers, each one establishing a connection with the other party's context manager and then transmitting the relevant data over the connection.

At this point in the protocol, since both parties possess the circuit structure, *CypherCircuit* enables the generating party to form a compact JSON representation of the garbled tables of ciphertexts and then transmit this serialized garbling to the evaluator. Along with these garbled tables of ciphertexts, the generator must also provide the evaluator with the circuit's input wire labels that correspond to the encoding of its own input. The package also facilitates the creation of a JSON representation of this encoding, which may be similarly transferred to the evaluator.

The exchange of circuit garbling and encoding information is followed by execution of oblivious transfer (OT) to furnish the circuit evaluator with input wire labels for values known only to them. *CypherCircuit* performs this OT protocol once for each input wire where a label was not sent by the circuit generator. The package follows the protocol of Even, Goldreich, and Lempel for general $\binom{2}{1}$OT, using the RSA cryptosystem for instantiating the public-private key operations. After completing OT, the evaluator has possession of every input wire label to the circuit. Like all other interactive procedures, the OT protocol is strictly executed by parties via their context managers.

Decoding the Garbled Circuit

With all of the input wire labels collected, the evaluating party may proceed to decode the circuit. Again, *CypherCircuit* follows the protocol laid forth by Kolesnikov and Schneider [66]. Using the collection of circuit input wire labels, the evaluator proceeds to decrypt truth tables for each gate in the circuit for which they possess the requisite input wire labels. For gate g_j, and input wire labels ℓ_a and ℓ_b (values potentially unknown), the evaluator can determine the value $H\left(\ell_a \parallel \ell_b \parallel j\right)$. Since the exclusive or (XOR) operation is its own inverse, the gate's output wire label ℓ_c may be calculated as

$$\ell_c = \mathfrak{e} \oplus H\left(\ell_a \parallel \ell_b \parallel j\right) \ , \tag{5.1}$$

where $\mathfrak{e}$ represents the ciphertext of ℓ_c encrypted symmetrically under ℓ_a and ℓ_b. Once all of the gates have been decrypted in this manner, the values on the circuit output wires $w_i \in W_O$ can be calculated from the equation

$$v_i = \mathfrak{e}_{v_i} \oplus H\left(\ell_i \parallel \text{`out'} \parallel i\right) \ . \tag{5.2}$$

It can be seen that Equations 5.1 and 5.2 are the inversions of Equations 3.7 and 3.8 respectively.

Trimming the Tedium

The *CypherCircuit* package was developed to provide an accessible tool for users without extensive cryptographic expertise. While this guiding philosophy has led to an emphasis on clarity and intuitiveness over sheer performance, it is recognized that building even remotely complicated circuits by hand is extremely tedious and error prone. Many functions (and even many operations) are non-obvious to implement as Boolean logic circuits. Because of this, the *CypherCircuit* package offers features to streamline the experience of a user building circuits so that they may execute more complicated functions. Keeping with the principle of accessibity, these features build upon the existing structure that has been previously defined—the basic `CircuitBoard` object along with circuit elements like `Wire` and gate-type objects.

At the most basic level, *CypherCircuit* provides a set of circuit components that perform frequently used operations. These components include basic comparators, adders, subtractors, multipliers, dividers, and similar atomic operations. In many cases, variations are provided to handle operations over single and multiple bits. These components can be added to an existing `CircuitBoard` object, connecting wires in the same way as the fundamental logic gates. In reality, these components are simply a collection of wires and gates that come preassembled for easy incorporation into a circuit. Only slightly more complex but following a similar pattern, *CypherCircuit* also provides several prefabricated circuits that compute entire functions. Like the built-in components, these "integrated circuits" are built entirely of wire and gate elements; however, unlike the components, each circuit consists of a complete *CypherCircuit* `CircuitBoard` and can be garbled, encoded, and decoded without requiring any other structural components.

Both of these prefabricated circuit structures still generally require that a user be proficient in understanding the designs of Boolean logic circuits. These types of constructions maintain easy access to the underlying circuit structures, but where *CypherCircuit* is aimed at users who are merely familiar with conceptual garbled circuits, it should not be required that a user be an expert in digital logic circuits any more than an expert in cryptography. In fact, this notion is prevalent in many published MPC protocols, which tend to expose interfaces that simplify circuit construction for end-users. As described in Section 3.5, these interfaces often entail compilation of a customized high-level programming language into a low-level Boolean circuit or by adapting existing languages to perform secure computations. What differentiates *CypherCircuit* is that it attempts to provide a high-level language for users to construct circuit-based functions, while preserving access to the individual circuit elements.

This objective is accomplished by leveraging Python's ability to overload operators. *CypherCircuit* defines a set of type-like objects (e.g. `CCInt`, `CCBool`, etc.) that represent a collection of wires defining the given type. Through operator overloading, *CypherCircuit* enables a user to manipulate instances of these types using traditional programming syntax, with operations on the type instances automatically constructing circuits. For integers, this results in common operations such as addition (`+`), multiplication (`*`), and comparisons (`>`,`<`, `==`) being directly translated by each type object into the corresponding adder, multiplier, or comparator components. These components are then directly added to the `CircuitBoard` by the type object.

Beyond operators, *CypherCircuit* also includes limited functionality for performing conditional

operations. The `CCInt` type provides a method that accepts three arguments: a condition (a `CCBool`) and two potential values (both `CCInt`). The function builds the circuitry to set the value of the returned `CCInt` object's wires to match one of the two potential input values depending on the value of the condition. This structure is similar to the one introduced in Obliv-C via the `obliv if` structure first mentioned in Section 4.2, though limitations of the Python language prohibit the *CypherCircuit* version from being integrated as seamlessly into the the development process.[1] Still, just like in Obliv-C, this setup forces *CypherCircuit* to evaluate both conditional blocks (zero and one) to avoid revealing private information related to the value of the condition. To only evaluate one block or the other according to the condition would potentially expose information about the chosen private inputs other than the function's output. Such knowledge would compromise the protocol's security.

From the bottom up, all of the structures that are designed to increase the usability of *CypherCircuit* are also implemented in a way that preserves user access to the framework's foundational machinery. Even when more sophisticated component or language constructions are included, users are always left with the ability to investigate and modify the circuit structure undergirding their calculation. Together, this practice ensures that *CypherCircuit* remains accessible, extensible, and transparent.

5.2.3 Optimizations

The traditional implementation of garbled circuits incurs high computational costs—often several expensive encryption operations per circuit gate. Because of this, optimizations are a key element in the practical implementation of garbled circuits in nearly every context. *CypherCircuit* also relies on optimizations to minimize the computational costs of building, exchanging, and garbling sophisticated circuits. Although computational performance is not nearly as high of a priority for *CypherCircuit*, it must still be considered in the package's design. For one thing, even a tool that otherwise satisfied the three guiding principles but which could not efficiently handle moderately sized calculations would see limited utility. Furthermore, where optimizations are ubiquitous across the garbled circuit (and greater MPC) landscape, it makes sense to include some of the most prevalent techniques in a package intended to facilitate examination of garbled circuit protocols.

Point-and-Permute

The first optimization included in *CypherCircuit* is the point-and-permute method. The theory behind the technique is discussed in Section 3.2.2, and *CypherCircuit* follows the point-and-permute procedure as it is outlined by Kolesnikov and Schneider [66]. As described in Section 5.2.2, the technique governs how tables of ciphertexts are shuffled, embedding information in each ciphertext that can be used to "point" to the ciphertext in the next gate's table that should be decrypted. Ul-

[1] The diverse catalog of external packages and modules available to Python may offer a more satisfactory substitute for programming oblivious conditionals in *CypherCircuit*, though an obvious candidate was not identified as of this writing.

timately no more than one ciphertext need be decrypted for each gate. *CypherCircuit* appends the permutation to the plaintext output label prior to encryption, resulting in a $\kappa + 1$ bit ciphertext.[2]

FreeXOR

To offer further improvements in efficiency, *CypherCircuit* also implements the FreeXOR technique described by Kolesnikov and Schneider [66]. As discussed in Section 3.2.2, gates that perform the XOR operation do not require garbling. When a global key offset represents the difference (modulo 2) between both labels on a wire, reliance on a random oracle is no longer necessary to eliminate correlation between encrypted rows in those XOR truth tables. Instead, the input wire labels can be simply used as one-time-pads to encrypt the corresponding output wire label (the property illustrated by Table 3.1).

When the FreeXOR technique is implemented, a table of ciphertexts is no longer required for the given `XOR` gate; rather, the output wire label can be determined directly from the input wires. *CypherCircuit* takes advantage of this property by using a special `FreeXorWire` class, derived from the standard `Wire` object, which stores a direct link to the previous input wires. Output wire labels on this `FreeXorWire` are generated on-demand by calculating the result from the two input wires. No ciphertext table is ever stored for an `XOR` gate. For users who are interested in performing computations without the FreeXOR optimization, this behavior can be deactivated.

"Smart" Wires

There are also many instances where circuits can reduce complexity by reducing redundancy or avoiding unnecessary calculations. One primary avenue for efficiency improvements is by intelligently handling wires that carry publicly known values. Sometimes these wires may be used for convenience in constructing circuit components (e.g. an adder component can be extended with a single fixed wire, which, when inverted, converts the component into a subtractor). Other times these wires represent public constants (e.g. a 2PC computation that evaluates the comparison $2x \stackrel{?}{=} y$ for the two parties' respective inputs x and y uses the constant 2). *CypherCircuit* provides a set of fixed wires—the `Hi` and `Lo` singleton wires—to represent wires that will *always* have a value of either zero or one. These wires are named to mimic their logical equivalents in a physical circuit; LO is equivalent to ground, and the single common terminal for each HI and LO wire on a physical circuitboard is a concept paralleled by their implementation in *CypherCircuit* as singletons.

The use of singleton fixed wires has a few significant efficiency implications. Most obviously, using singleton `Hi` and `Lo` wires reduces the memory footprint of a garbled circuit when user applications call for multiple fixed wire inputs of the same type. These wires do not need to be duplicated for structural or security reasons, and so consolidating them saves space in system memory, along

[2]Having the ciphertext be $\kappa + 1$ bits preserves security at the level set by the security parameter before the point-and-permute method was introduced. This is a non-traditional implementation according to Ball, Malkin, and Rosulek [79], and it also incurs a performance penalty; however, it was perceived that reducing the effective security parameter without user input would be poor security practice, even if largely inconsequential.

with the time needed to generate and label them separately. Rather than casting the burden of efficiency management onto the users, *CypherCircuit* simply recognizes the occurrence of fixed wires and returns the singleton case.

There are other less obvious efficiency gains that can be achieved through this type of intelligent circuit construction as well. Returning to the case of fixed wires, there are situations where a fixed wire input can propagate through a circuit, resulting in numerous unnecessary calculations if not otherwise addressed. Take, for example, the case where a fixed `Lo` wire feeds into an `AND` gate. Where the value of `Lo` is always zero, the output wire of the `AND` gate will also always be zero. In this case, *CypherCircuit* will analyze the truth table of potential outputs from a gate, identify that the only potential output wire value is zero, and automatically reassign that gate's output wire to be the singleton `Lo` wire. Recalling the extensibility design principle, *CypherCircuit* relies on the gate's truth table to make these identifications so that these assessments can be generalized to any conceivable type of gate.

It is not only fixed wires that benefit from this style of assessment, however. `NOT` gates in *CypherCircuit* are similarly able to recognize when they are given a wire that is already the inverse of another, and they return the existing inverse rather than create a new redundant wire. `XOR` gates do the same, recognizing that an XOR operation between fixed wires, a duplicate wire input, or a pair of wire inverses all produce predictable outcomes. For fixed wires, any `XOR` gate that receives a `Lo` wire input will always output the other input wire unaltered, and any `XOR` gate that receives a `Hi` input wire will always output the inverse of the other input wire. When a single wire is selected as both inputs, the output of the `XOR` gate will always be a `Lo` wire; when a pair of inverse wires are selected as inputs, the output of the `XOR` gate will always be fixed as `Hi`.

Although these cases superficially appear to be edge cases, experience developing *CypherCircuit* suggests that they are not entirely uncommon. When building circuits using more sophisticated components (such as adders, subtractors, multipliers, and dividers) relationships between inter-component wires become complicated. While managing construction in this way adds complexity to the *CypherCircuit* software, it does so to make the package more intuitive—avoiding unnecessary duplication—while also enhancing performance.

5.3 Comparisons to Existing MPC Frameworks

The *CypherCircuit* package was not designed to match the speed of contemporary MPC frameworks, instead focusing on providing an intuitive look into the garbled circuit protocol. Even so, it is useful to document how the *CypherCircuit* package compares to other modern and state-of-the-art codes when discussing its development.

To provide a comparison, the *CypherCircuit* package, Obliv-C library [99], and MP-SPDZ [95] framework were all used to execute a series of benchmark tests. The benchmark used here was based on the Euclidean distance function, with programs created in all three frameworks to calculate the distance between a pair of N-dimensional vectors. All programs coordinated their MPC protocols between two parties connected locally via a TCP network, and were run in Docker containers to ensure a consistent execution environment and future reproducibility. The average calculation time

Table 5.1: The average runtime (and standard deviation σ) over ten tests of MPC programs written in *CypherCircuit*, Obliv-C, and MP-SPDZ to calculate the Euclidean distance between two N-dimensional vectors.

	CypherCircuit		**Obliv-C**		**MP-SPDZ** Binary (Yao)		**MP-SPDZ** MASCOT	
N**-dimensions**	**Time (s)**	σ **(s)**	**Time (s)**	σ **(s)**	**Time (s)**	σ **(s)**	**Time (s)**	σ **(s)**
1	16.52	1.07	0.65	0.08	–	–	1.29	0.05
2	28.79	1.62	0.63	0.01	–	–	1.27	0.02
3	44.96	1.78	0.69	0.09	–	–	1.27	0.02
4	59.23	4.00	0.66	0.05	–	–	1.27	0.01
5	72.13	4.51	0.64	0.02	–	–	1.27	0.01
10	144.52	6.68	0.61	0.04	–	–	1.29	0.02
50	749.46	28.11	0.64	0.02	–	–	1.30	0.01
100	1502.21	38.48	0.67	0.01	1.20	0.10	1.32	0.02

for each trial, determined using ten repetitions of the trial using a randomized vector, is displayed in Table 5.1. All calculations were run on a commercially available, off-the-shelf personal computer with a 2.3 GHz 8-Core Intel Core i9 processor and 16 GB memory. The Docker containers used for the trials of both frameworks were built on an image of Ubuntu 20.04, set using all of the default parameters. All containers were permitted access to at least 4 GB of memory.

It should also be noted here that this benchmark was intended solely to compare the performance of *CypherCircuit* to other modern MPC solutions in a *basic* example, and does not attempt to leverage any specific enhancements, optimizations, or design features that may improve the performance of either existing framework. As such, the times recorded do not necessarily represent the optimal runtimes that could be achieved using either of those two frameworks.

For consistency, benchmarks for all three frameworks were originally designed to use binary garbled circuits (Yao's protocol). Binary circuits are the only protocol that is supported by *CypherCircuit* and Obliv-C, and it is a protocol that is included in the MP-SPDZ package. Unfortunately, it was found that MP-SPDZ was unable to compile binary circuits that accepted public inputs known to all parties provided at runtime, even though this was a nominally provided functionality. MP-SPDZ, as a state-of-the-art tool, is still under active development and may gain this capability in the future; however, a publicly provided input value was required to set the vector length in each set of benchmark trials. Instead, since MP-SPDZ allows compilation with a variety of different protocols, the benchmark was compiled and evaluated using the MASCOT protocol [85]. Although these MASCOT results should not be directly compared to either those from *CypherCircuit* or Obliv-C (because the MASCOT protocol uses arithmetic circuits and malicious security while the other two codes only use binary circuits and semi-honest security), the execution timing for MP-SPDZ's

MASCOT protocol gives a rough comparison of how the framework might compare to the other two. For the sake of providing a more useful comparison, a single vector length of 100 elements was also hardcoded into the MP-SPDZ binary circuit benchmark script and evaluated. This singular result is shown in the first pair of MP-SPDZ columns in Table 5.1.

Otherwise, the most obvious difference between the three frameworks is the magnitude of execution time, though it is unsurprising that *CypherCircuit* is significantly slower than either of the other two frameworks. First and foremost, *CypherCircuit* is built using Python and in a structure designed for human intuition and user exploration. Therefore each circuit object (wires, gates, tables, etc.) are all Python objects requiring memory. While attempts have been made to consolidate the memory footprint of these objects as much as possible, providing a consistent and accessible interface limited flexibility in restructuring the code for enhanced performance. Second, optimization techniques such as oblivious transfer extension remain to be included into *CypherCircuit*. Third, whereas programs written in Obliv-C and MP-SPDZ are compiled, *CypherCircuit* circuits written in Python are interpreted at runtime and are unable to leverage the enhanced execution times of compiled code.[3]

Another notable difference in the software frameworks is apparent when considering the scaling of the codes. The runtimes of the distance calculations performed using *CypherCircuit*, and to a less visible extent MP-SPDZ, predictably scale larger with increasing vector length. The scaling for *CypherCircuit* is linear, as more squared differences between pairs of vector elements are summed, the framework creates the same number of equal sized subcircuit components. Obliv-C, however, does not appear to scale upward (in fact, the difference between the runs is largely insignificant). This is an artifact of how the executables designed to perform the Euclidean distance calculation in Obliv-C handle memory. Imitating the numerous examples offered by the library, the Obliv-C executables were written to allocate a fixed block of memory in advance, which is never exceeded by any of the vector sizes used in this problem. With longer vector pairs (e.g. 1000 elements), the Obliv-C and MP-SPDZ libraries see far more dramatic increases in computation time.

These results make clear that while *CypherCircuit* is designed to serve as an educational demonstration tool, sophisticated safeguards calculations like those performed in Chapter4 will certainly require a more efficient production grade software solution.

5.4 Feedback and Future Direction

The *CypherCircuit* package is intended to be an intuitive tool for non-expert audiences to gain an understanding of the garbled circuit protocol. With the goal of user accessibility in mind, a team of users at Oak Ridge National Laboratory (ORNL) who were familiar with MPC protocols—though not cryptographic experts by training—was enlisted to review the package. On the whole, the review team confirmed that the package was both intuitive and accessible, validating efforts towards meeting those elements of the design principles outlined in Section 5.2. The review provided a

[3] *CypherCircuit* does make use of Cython for certain calculation components to improve the size and speed of certain aspects of the garbled circuit protocol.

glimpse of how users are likely to approach using the package and offered insight to where future development efforts should be concentrated.

Among the suggestions provided by the review team, the majority focused on improving the user experience when handling large calculations. It was shown in Section 5.3 that moderately complicated *CypherCircuit* programs may take several minutes or more to complete, and updating users throughout this time is important. Specific measures for enhancing the code could include adding progress indicators for actions such as building, garbling, or transferring circuit components; implementing more robust error handling and a checkpoint system for resilience against interrupted network connections; and designing a system to partition large circuits into smaller, more manageable subcircuits that could be evaluated separately (and potentially facilitate parallelization).

Another suggestion provided by the review team was to provide easy access to a comprehensive overview of each circuit, such as a single circuit summary detailing the number of wires and gates in a circuit, as well as information regarding any particular optimizations (e.g. *FreeXOR*) that have been selected. While all of this information is already accessible to a user via the Python API, an overview summary system could be implemented to facilitate direct, easy access.

Finally, while it is obvious from the results presented in Section 5.3 that *CypherCircuit* is indeed far slower than Obliv-C, it should be noted that several optimizations could still potentially be implemented to improve the package's efficiency. Techniques like the half-gates method [120] or row reduction [81] could be added to reduce the number of encryption operations needed for any given circuit. Similarly, adding some form of OT extension [121, 122] to the transfer process for input wire labels would dramatically reduce the computational cost of the expensive and repetitive public/private key encryption required by the *CypherCircuit* package's current OT implementation.

In general, however, the ORNL review was overwhelmingly positive, and suggests that *CypherCircuit* is a suitable tool for users seeking to build an understanding of garbled circuit protocols. From this reception, it is encouraging to consider that the tool may indeed be useful for demonstrating the application of MPC in novel applications—especially those where the participants may be skeptical of the validity of such an algorithm, like in the case of international nuclear safeguards.

Chapter 6

Conclusion and Discussion

It is in the interests of the international community to prevent the proliferation of nuclear weapons, a stance that is affirmed by the near unanimous support for the Treaty on the Non-Proliferation of Nuclear Weapons (NPT) among the nations of the world. Such a goal is a ceaseless endeavor, and one that requires cooperation between as many nation states as possible. In increasingly tense and polarized global environments, international partnerships are likely to prove more and more difficult to build and maintain, especially when negotiated between rival parties. Globally pursued compliance and enforcement of nuclear nonproliferation standards is almost certainly always going to pose a challenge, but becomes even more difficult as claims of sovereignty outweigh the perceived benefits of international collaboration towards common goals. Whether bred from nationalistic or defensive postures, these fiercely defended notions of sovereignty have the potential to drive States to question or attempt to skirt commitments to abide by international guidelines. In this setting, trust between members of the international community is likely to become a thinly stretched commodity.

Environments like these are where privacy-preserving algorithms present tremendous opportunities. By enabling two or more factions—nuclear facilities and their regulators; States and global agencies—to engage in monitoring that is both valuable in its conclusions and intrinsically secure in its implementation, the level of trust required between parties can be reduced. The international community can trust that safeguards are being administered robustly, a State or nuclear facility can trust that its data is being protected, and neither needs to place that trust in the other as they operate using a trusted algorithm that is provably secure.

The implementation of such technological advances has been proposed and demonstrated in this dissertation, such that the potential benefits of leveraging multiparty computation (MPC) for nuclear safeguards might be considered in future diplomatic negotiations and policy making. Nuclear safeguards currently require that the International Atomic Energy Agency (IAEA) gain access to any and all data that they wish to analyze, but this work provides the first demonstration of how MPC could be applied to safeguards datasets, preserving the privacy of a nuclear facility's data while simultaneously bolstering IAEA analyses.

6.1 Summary

After reviewing the two distinct fields of nuclear safeguards and privacy-preserving computation, Chapter 4 offered two demonstrations of how MPC might be used in a safeguards context. Using a large dataset of authentic radiation spectra, collected as a time series over two months by the Modeling Urban Scenarios and Experiments (MUSE) project, these demonstrations introduce MPC as a potential solution to challenges encountered in the safeguards community, and serve as a proof-of-concept for future work incorporating privacy-preserving computational techniques into nuclear safeguards protocols. On a wider scale, this work is believed to represent the first known implementation of MPC intended for time series anomaly detection that is designed to be performed on-line, at or near real-time.

In both of the demonstration trials described in Chapter 4, garbled circuits were prepared to execute anomaly detection protocols that preserve the privacy of a nuclear facility as it interacts with a regulator. Both algorithms featured techniques based on a prediction model to identify single outlier data points among the input radiation spectra. As expected, fast and uncomplicated algorithms like the gross counting method used in the first trial show promise in identifying potential outliers quickly; the true material transfer being sought was flagged by the algorithm, although not without the presence of false positive signals in the analysis. More sophisticated algorithms, like the region of interest (ROI) calculation derived from spectral comparison ratios (SCRs) used in the second trial, show a far stronger ability to discriminate potential material diversions from background. When analyzing the MUSE data, even a simplistic three region calculation based on a single known prior transfer event was able to identify and flag two subsequent material transfer events without any false positive readings. Even with this success, it is certain that the ability of similar algorithms to successfully find signals depends strongly on the dataset properties and information available to the two parties. Lower quality data streams may obscure easy observation and comparison, making it more difficult to draw meaningful conclusions from anomaly detection algorithms. Likewise, stricter MPC security requirements may slow algorithms to the point where only the simplest anomaly detection techniques are practical choices for implementation. Regardless of those possible hurdles, however, this work was able to show that anomaly detection analyses of this nature could be performed in a privacy-preserving manner. The fact that MPC could be viable in a safeguards environment is plainly evident.

It is important to emphasize here that despite their promise, MPC solutions to safeguards problems *must* be investigated thoroughly on a case-by-case basis before adoption into a safeguards program. As it was noted in Chapter 4, these techniques are not without limitation. For one, all of the demonstrations provided here were performed under the semi-honest security assumption—namely that all parties were obeying the protocols and executing the garbled circuit as expected. This assumption would not be appropriate in a safeguards context where systems must be designed to be resilient in the face of adversarial manipulation. However, malicious security is an active area of MPC research and MPC techniques that incorporate malicious security are readily available. The use of the semi-honest, passive security model in this work does not preclude the use of malicious, active security models in production level applications, and the proof-of-concept remains valid.

Beyond the active security of the MPC algorithm, a skeptical safeguards administrator would

certainly point to the algorithm's inability to prevent a malicious participant from feeding fabricated data into the algorithm as a severe weakness. This spoofed data would be kept private, and therefore be far more difficult to recognize as falsified. There are a few potential solutions to this dilemma.

One potential solution is for the regulator to know enough about what the facility input should look like to avoid egregious misrepresentations from being fed into the algorithm. Then, using that knowledge along with an intelligent choice of its own private input parameters, the regulator could make it difficult for a facility to spoof input data. Although some privacy might be compromised here (offering the regulator even limited knowledge of the input would still require some sacrific of privacy on the part of a facility), the prior set of known information could act as a sort of template for expected behavior while preserving the privacy of the finer nuances of the facility data. Ideally, in such a compromise, any up-front forfeitures of privacy could be reduced to the point of negligibility when compared to the scope of the privacy-preserving monitoring.

Alternatively, a facility could be forced to commit to its input data stream in such a way that if a regulator later had questions about a certain time period, the facility could be compelled to provide the historical data. For example, an otherwise highly reliable MPC solution might be paired with more traditional sets of procedures that are defined by current safeguards protocols and mandated under existing safeguards agreements. Then, in the event that a question arose from the traditional safeguards information set, a regulator could request access to the input data to the MPC algorithm. They could validate that the data matched their expectations *and* and could confirm that it matched the facility's previous commitment.

Recent work—including efforts on an accompanying component to this project—has investigated the applicability of distributed ledger technology (DLT) to this commitment challenge. In such a distributed system, cryptographic hashes of input values are stored in a set of shared memory locations (perhaps one ledger per IAEA Member State). By leveraging the data immutability properties of DLT, a party that provides input cannot later deny having supplied that data without having had to convince some significant fraction of parties to abet its forgery.

Assuming a strong commitment scheme along these lines, fabricated data would be revealed either when provided to inquiring inspectors or when a passable set of fabricated data failed to match the commitment for data that was provided to the original MPC algorithm. Such an option might reduce the privacy of the algorithm, but perhaps only within a reasonable set of circumstances that would be tolerable to the concerned parties. A transparent process for conducting audits in this fashion would ideally prevent a regulator from abusing their authority to expose more private information than absolutely necessary.[1]

The driving motivation behind implementing these extra protocols correctly is so important that it bears repeating—any MPC implementation of safeguards processes *must* be verified as secure on a case-by-case basis. Any method that introduces an insecure algorithm adds increased risk of being manipulated into providing conclusions offering a false sense of security even in the presence of other alarming data. In that case, a weak auxiliary MPC system is worse than no system at all.

[1] A parallel may be drawn here to search warrants issued in the United States. Search and seizure is generally protected against under the Fourth Amendment of the U.S. Constitution, but court ordered warrants can supercede this protection. Most would likely agree, however, that the Fourth Amendment still serves as a valuable check against institutional overreach for protecting individual privacy.

Moving on, it was also recognized in this work that the tools available to general audiences for testing, understanding, and prototyping MPC protocols were scarce. The majority of MPC software tools are designed to showcase performance improvements of new techniques or facilitate comparisons between the many various MPC implementations. More recently, several codes have attempted to make MPC calculations accessible to general audiences and practictioners without MPC expertise; however, even these frameworks do not emphasize transparency of the underlying calculation. In applications like nuclear safeguards, where prior experience with MPC is severeley limited and parties may be inherently skeptical of the security of a MPC tool, it is essential to use tools that can offer clear and convincing demonstrations of MPC execution. It was for this reason that this project also included the development of the *CypherCircuit* software framework, introduced in Chapter 5.

CypherCircuit was designed to offer a clear, intuitive, transparent tool for gaining experience with MPC—specifically with garbled circuits. It emphasizes the user experience, preserving the ability for users to interact directly with the circuits they create to perform their calculations. With the package, practictioners may have a hand in circuit design and protocol execution while also easily exploring a well-documented and rigorously tested code base. Most importantly, despite its performance penalty when compared to state-of-the-art MPC frameworks, the *CypherCircuit* package is designed to facilitate an understanding of MPC among its users. This aspect is an absolutely critical aspect when attempting to push adoption of MPC technologies in high-risk fields such as nuclear safeguards.

Together, the two demonstration trials presented in Chapter 4 and the *CypherCircuit* package introduced in Chapter 5 are intended to showcase that MPC is not simply a notional, abstract element of safeguards science fiction, but an emerging technology that should be considered when building the safeguards regimes of the near future.

6.2 Future Work

Where this dissertation served as the introduction of MPC to the field of nuclear safeguards, there are countless avenues for additional exploration regarding where, when, and how to incorporate privacy-preserving computational techniques into the international nuclear inspection landscape. Of particular interest would be a determination of which anomaly detection algorithms (and which flavors of MPC) would be best suited to particular safeguards problems. This study presented just two anomaly detection algorithms, both coded as garbled circuits, without conducting an exhaustive search for either radiation detection techniques or MPC implementations that might prove superior in real—rather than simulated—safeguards applications. Some effort has already been devoted to exploring enhanced applications extending this work, for instance using state-of-the-art anomaly detection techniques like grammar compression to reduce the quantities of data that must be processed by a garbled circuit. The grammar compression algorithms are designed to efficiently seek out anomalies in large datasets, a technique that could conveniently offset the efficiency losses introduced by privacy-preserving algorithms. Such integration presents its own challenges, however, which are not covered here but are to be addressed in future publications.

MPC has other potential applications within the scope of nuclear nonproliferation, albeit distinct from nuclear safeguards. For example, MPC could have benefits for those interested in augmenting the capabilities of arms control verifiers. Philippe et al. [123] have already proposed using physical zero-knowledge proofs (ZKPs) to perform arms control verification without exposing classified State information to a treaty verifier. Returning to the digital, cryptographic roots of that technology, it is not farfetched to believe that other digital arms control verificiation technology could leverage MPC to achieve similar objectives: keeping input data secure while offering valuable insights into the contents of a weapons vessel to corrobarate compliance with arms control and reduction agreeements.

Even outside the nuclear non-proliferation landscape, but within the nuclear industry, MPC could offer new avenues of collaboration. Civilian nuclear reactor operators are often highly protective of their data, especially those motivated out of concern for protecting proprietary information and preserving their competitive edge by keeping tightly held trade secrets. At the same time, the nuclear industry at large would benefit if these reactor operators were to be willing and able to share more operational details amongst themselves. MPC offers one way that these operators might be able to come together to learn from the cumulative experience of the industry without compromising their own respective collections of operational data.

Ultimately, MPC represents an opportunity for safeguards administrators. The technology has matured sufficiently in the past two decades to be a practical consideration when designing safeguards systems, and the privacy guaranteed by the algorithms is likely to be an increasingly attractive option for States concerned about their own privacy and the security of their data. This is especially true in a digitally connected world that feels increasingly vulnerable.

MPC is unlikely to be a technology that will upend the current safeguards landscape, nor should it be. Existing technologies should instead be augmented by including MPC in the suite of tools available to safeguards regulators. No safeguards technology or practice will ever be totally insurmountable to every adversary, but when taken together, a comprehensive safeguards regime will prevent, deter, and detect malicious activity in all but a vanishingly small number of cases. In this spirit, using MPC techniques to bolster existing safeguards offers the potential to significantly increase the coverage of nuclear safeguards and reassure the global community that it remains safe from the risk of nuclear proliferation.

References

[1] D. D. Eisenhower. *Address Before the General Assembly of the United Nations on Peaceful Uses of Atomic Energy, New York City*. [Accessed 7 October 2020]. Dec. 1953.

[2] D. Holloway. "Nuclear weapons and the escalation of the Cold War, 1945–1962". *The Cambridge History of the Cold War*. Ed. by M. P. Leffler and O. A. Westad. Vol. 1. The Cambridge History of the Cold War. Cambridge University Press, 2010, pp. 376–397. DOI: 10.1017/CHOL9780521837194.019.

[3] U.S. Department of Energy Office of History and Heritage Resources. *Espionage and the Manhattan Project*. [Accessed 6 November 2021]. 2005.

[4] G. A. Goncharov and L. D. Ryabev. "The development of the first Soviet atomic bomb". *Physics-Uspekhi* 44.1 (Jan. 2001), pp. 71–93. DOI: 10.1070/pu2001v044n01abeh000875.

[5] A. Goldberg. "The Atomic Origins of the British Nuclear Deterrent". *International Affairs (Royal Institute of International Affairs 1944–)* 40.3 (July 1964), pp. 409–429. DOI: 10.2307/2610825.

[6] H. M. Kristensen and R. S. Norris. "Global Nuclear Stockpiles, 1945–2013". *Bulletin of the Atomic Scientists* 69.5 (2013), pp. 75–81. DOI: 10.1177/0096340213501363.

[7] International Atomic Energy Agency. *From Obninsk Beyond: Nuclear Power Conference Looks to Future*. June 2004.

[8] American Society of Mechanical Engineers. *A National Historic Mechanical Engineering Landmark: Experimental Breeder Reactor I*. June 1979.

[9] U.S. Nuclear Regulatory Commission. "Nuclear Regulatory Legislation". NUREG-0980. Dec. 2015. Chap. 2.

[10] J. W. Finney. "U.S. Will Dedicate Atom Plant Today". *The New York Times* (May 1958).

[11] W. G. Weart. "Eisenhower Hails Atoms for Peace". *The New York Times* (May 1958).

[12] *Statute of the International Atomic Energy Agency*. 1957.

[13] International Atomic Energy Agency. *INFCIRC/153: The Structure and Content of Agreements between the Agency and States Required in Connection with the Treaty on the Non-Proliferation of Nuclear Weapons*. Vienna, Austria, June 1972.

[14] U.S. Nuclear Regulatory Commission. *Nuclear Security and Safeguards*. [Accessed 10 October 2020]. Mar. 2020.

[15] International Atomic Energy Agency. *INFCIRC/140: Treaty on the Non-Proliferation of Nuclear Weapons: Notification of entry into force*. Vienna, Austria, Apr. 1970.

[16] E. M. Chossudovsky. "The Origins of the Treaty on the Non-Proliferation of Nuclear Weapons: Ireland's Initiative in the United Nations (1958–61)". *Irish Studies in International Affairs* 3.2 (1990), pp. 111–135.

[17] U.S. Delegation to the 2010 Nuclear Nonproliferation Treaty Review Conference. *Treaty on the Non-Proliferation of Nuclear Weapons*. 2010.

[18] Danlaycock Allstar86 L.tak. *Participation in the Nuclear Non-Proliferation Treaty*. Image. [Accessed 29 April 2021]. Feb. 2014.

[19] *Tratado para la Proscripción de las Armas Nucleares en la América Latina y el Caribe*. [Accessed 5 November 2021]. Mexico City, Mexico, June 2018.

[20] *South Pacific Nuclear Free Zone Treaty*. [Accessed 5 November 2021]. Rarotonga, Cook Islands, Aug. 1985.

[21] *African Nuclear Weapon Free Zone Treaty*. [Accessed 5 November 2021]. Cairo, Egypt, Apr. 1996.

[22] *Treaty on a Nuclear-Weapon-Free Zone in Central Asia*. [Accessed 5 November 2021]. Semipalatinsk, Kazakhstan, Sept. 2006.

[23] *Treaty on the Southeast Asia Nuclear Weapon-Free Zone*. [Accessed 5 November 2021]. Bangkok, Thailand, Dec. 1995.

[24] International Atomic Energy Agency. *INFCIRC/395: Agreement Between the Republic of Argentina and the Federative Republic of Brazil for the Exclusively Peaceful Use of Nuclear Energy*. Vienna, Austria, Nov. 1991.

[25] L. Rockwood. *The International Nuclear Nonproliferation Regime*. Audio. [Accessed 28 April 2021]. Aug. 2020.

[26] *IAEA Safeguards Glossary*. Vienna, Austria: International Atomic Energy Agency, 2002.

[27] International Atomic Energy Agency. *Guidance for States Implementing Comprehensive Safeguards Agreements and Additional Protocols*. IAEA Services Series 21. Vienna, Austria, May 2016.

[28] U.S. Nuclear Regulatory Commission. *Nuclear Material Control and Accounting*. [Accessed 5 November 2021]. Oct. 2021.

[29] A. Thompson and B. N. Taylor. *Guide for the Use of the International System of Units (SI)*. NIST Special Publication 811. Gaithersberg, Maryland, USA: National Institutes of Standards and Technology, Mar. 2008.

[30] J. J. Duderstadt and L. J. Hamilton. *Nuclear Reactor Analysis*. New York, New York, USA: John Wiley & Sons, Inc., 1976.

[31] National Nuclear Data Center. *NuDat 2.8*. [Accessed 20 October 2020].

[32] K. S. Krane. *Introductory Nuclear Physics*. USA: John Wiley & Sons, Inc., 1988.

[33] M. Wang et al. "The AME2016 atomic mass evaluation". *Chinese Phys. C* 41.3 (Mar. 2017). DOI: 10.1088/1674-1137/41/3/030003.

[34] S. Glasstone and P. J. Dolan. *The Effects of Nuclear Weapons*. United States Department of Defense, United States Department of Energy, 1977.

[35] A. Pritzker and W. Hälg. "Radiation dynamics of a nuclear explosion". *Journal of Applied Mathematics and Physics (ZAMP)* 32 (Jan. 1981), pp. 1–11. DOI: 10.1007/BF00953545.

[36] C. W. Forsberg, C. M. Hopper, and ... *Definition of Weapons-Usable Uranium-233*. Tech. rep. ORNL/TM-13517. Oak Ridge National Laboratory, Mar. 1998.

[37] International Atomic Energy Agency. *INFCIRC/540: Model Protocol Additional to the Agreement(s) Between State(s) and the International Atomic Energy Agency for the Application of Safeguards*. Vienna, Austria, Sept. 1997.

[38] International Atomic Energy Agency. *Nuclear Material Accounting Handbook*. IAEA Services Series 15. Vienna, Austria, May 2008.

[39] I. Tsvetkov et al. "Implementation of Short Notice Random Inspection (SNRI) at Japanese Low Enriched Uranium (LEU) Bulk Facilities–The Experience Gained and an Inspectorate Perspective". *Symposium on International Safeguards*. IAEA, 2001.

[40] H.-D. Choi and Kim J. "Basic characterization of uranium by high-resolution gamma spectroscopy". *Nuclear Engineering and Technology* 50.6 (Aug. 2018), pp. 929–936. DOI: 10.1016/j.net.2018.04.008.

[41] D. Calma. *Gamma Spectrometer*. IAEA. [Accessed 27 October 2021] Used under conditions of the CC BY-NC-ND 2.0 license. Apr. 2015.

[42] International Atomic Energy Agency. *Safeguards Techniques and Equipment: 2011 Edition*. International Nuclear Verificiation Series 1. Vienna, Austria, 2011.

[43] International Atomic Energy Agency. "Non-Destructive Assay: Instruments and Techniques for Agency Safeguards". *IAEA Bulletin* 19.5 (Oct. 1977).

[44] L. E. Smith and A. R. Lebrun. "Design, Modeling and Viability Analysis of an Online Uranium Enrichment Monitor". *2011 IEEE Nuclear Science Symposium Conference Record*. 2011, pp. 1030–1037. DOI: 10.1109/NSSMIC.2011.6154314.

[45] V. Fournier and M. Gaspar. *New IAEA Uranium Enrichment Monitor to Verify Iran's Commitments under JCPOA*. News release. Jan. 2016.

[46] S. Nakagawa et al. *Development of the Unattended Spent Fuel Flow Monitoring Safeguards System (UFFM) for the High Temperature Engineering Test Reactor (HTTR) (Joint Research)*. Tech. rep. Japan Atomic Energy Agency, Feb. 2007.

[47] D. Calma. *IAEA Seals*. IAEA. [Accessed 4 November 2021] Used under conditions of the CC BY-SA 2.0 license. Jan. 2005.

[48] D. Calma. *COBRA fibre optic seal*. IAEA. [Accessed 4 November 2021] Used under conditions of the CC BY-SA 2.0 license. Jan. 2005.

[49] D. Calma. *XCAM Safeguards Surveillance Camera*. IAEA. [Accessed 30 April 2021] Used under conditions of the CC BY-NC-ND 2.0 license. Mar. 2015.

[50] D. Calma. *Safeguards Surveillance System Server*. IAEA. [Accessed 7 November 2021] Used under conditions of the CC BY-NC-ND 2.0 license. Mar. 2015.

[51] K. Tolk, P. Merkle, and M. Aparo. "The Impact of Safeguards Authentication Measures on the Facility Operator". *8th International Conference on Facility Operations – Safeguards Interface*. SAND2008-2065C. Portland, Oregon, USA, Mar. 2008.

[52] M. A. Thomas et al. "Field trial of the enhanced data authentication system (EDAS)". *ESARDA Bulletin* 54 (June 2017).

[53] A. Shamir, R. L. Rivest, and Adleman L. M. *Mental Poker*. Tech. rep. MIT/LCS/TM—l25. MIT, Jan. 1979.

[54] A. Shamir. "How to Share a Secret". *Communications of the ACM* 22.11 (Nov. 1979), pp. 612–613. DOI: 10.1145/359168.359176.

[55] O. Goldreich, S. Micali, and A. Wigderson. "How to Play ANY Mental Game". *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[56] A. C. Yao. "Protocols for secure computations". *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. 1982, pp. 160–164.

[57] A. C. Yao. "How to Generate and Exchange Secrets". *27th Annual Symposium on Foundations of Computer Science* (Toronto, Ontario, Canada). IEEE, Oct. 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25.

[58] O. Goldreich. "Cryptography and cryptographic protocols". *Distributed Computing* 16 (Sept. 2003), pp. 177–199. DOI: 10.1007/s00446-002-0077-1.

[59] M. Bellare, V. T. Hoang, and P. Rogaway. "Foundations of Garbled Circuits". *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, pp. 784–796. DOI: 10.1145/2382196.2382279.

[60] D. Beaver, S. Micali, and P. Rogaway. "The Round Complexity of Secure Protocols". *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*. STOC '90. Baltimore, Maryland, USA: Association for Computing Machinery, 1990, pp. 503–513. DOI: 10.1145/100216.100287.

[61] O. Goldreich. *Foundations of Cryptography. II: Basic Applications*. Vol. 2. Cambridge University Press, May 2004. DOI: 10.1017/CBO9780511721656.

[62] A. Beimel and B. Chor. "Universally Ideal Secret Sharing Schemes (Preliminary Version)". *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '92. Santa Barbara, California, USA, Aug. 1992, pp. 183–195.

[63] D. Evans, V. Kolesnikov, and M. Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*. NOW Publishers, 2018.

[64] Y. Lindell. "Secure Multiparty Computation". *Communications of the ACM* 64.1 (Dec. 2020), pp. 86–96. DOI: 10.1145/3387108.

[65] Y. Aumann and Y. Lindell. "Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries". *Theory of Cryptography*. Ed. by Salil P. Vadhan. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, Feb. 2007, pp. 137–156. DOI: 10.1007/978-3-540-70936-7_8.

[66] V. Kolesnikov and T. Schneider. "Improved Garbled Circuit: Free XOR Gates and Applications". *Automata, Languages and Programming*. Ed. by L. Aceto et al. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, 2008, pp. 486–498.

[67] B. Pinkas et al. "Secure Two-Party Computation Is Practical". *Advances in Cryptology – ASIACRYPT 2009*. Ed. by M. Matsui. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, Dec. 2009, pp. 250–267. DOI: 10.1007/978-3-642-10366-7_15.

[68] Y. Huang et al. "Faster Secure Two-Party Computation Using Garbled Circuits". *Proceedings of the 20th USENIX Conference on Security*. SEC'11. San Francisco, California, USA: USENIX Association, 2011, p. 35.

[69] P. S. Nordholt et al. *State of the Art Analysis of MPC Techniques and Frameworks*. Tech. rep. 731583. Scalable Oblivious Data Analytics, Sept. 2017.

[70] S. Gueron et al. "Fast Garbling of Circuits Under Standard Assumptions". *Journal of Cryptography* 31.3 (July 2018), pp. 798–844. DOI: 10.1007/s00145-017-9271-y.

[71] E. M. Songhori et al. "ARM2GC: Succinct Garbled Processor for Secure Computation". *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. Las Vegas, Nevada, USA: Association for Computing Machinery, June 2019. DOI: 10.1145/3316781.3317777.

[72] S. Even, O. Goldreich, and A. Lempel. "A Randomized Protocol for Signing Contracts". *Communications of the ACM* 28.6 (June 1985), pp. 637–647. DOI: 10.1145/3812.3818.

[73] M. O. Rabin. *How To Exchange Secrets with Oblivious Transfer*. Tech. rep. TR-81. Harvard University, May 1981.

[74] M. Naor and B. Pinkas. "Oblivious Transfer and Polynomial Evaluation". *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*. STOC '99. Atlanta, Georgia, USA: Association for Computing Machinery, May 1999, pp. 245–254. DOI: 10.1145/301250.301312.

[75] W. Aiello, Y. Ishai, and O. Reingold. "Priced Oblivious Transfer: How to Sell Digital Goods". *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*. EUROCRYPT '01. Berlin, Heidelberg: Springer-Verlag, May 2001, pp. 119–135.

[76] S. Laur and H. Lipmaa. "A New Protocol for Conditional Disclosure of Secrets and Its Applications". *Proceedings of the 5th International Conference on Applied Cryptography and Network Security*. ACNS '07. Zhuhai, China: Springer-Verlag, June 2007, pp. 207–225. DOI: 10.1007/978-3-540-72738-5_14.

[77] A. Ben-Efraim. "On Multiparty Garbling of Arithmetic Circuits". *Advances in Cryptology – ASIACRYPT 2018*. Ed. by T. Peyrin and S. Galbraith. Springer International Publishing, Oct. 2018, pp. 3–33. DOI: 10.1007/978-3-030-03332-3_1.

[78] A. Dupin, D. Pointcheval, and C. Bidan. "On the Leakage of Corrupted Garbled Circuits". *Provable Security*. Ed. by J. Baek, W. Susilo, and J. Kim. Jeju, South Korea: Springer International Publishing, Oct. 2018, pp. 3–21. DOI: 10.1007/978-3-030-01446-9_1.

[79] M. Ball, T. Malkin, and M. Rosulek. "Garbling Gadgets for Boolean and Arithmetic Circuits". *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria). CCS '16. New York, New York, USA: Association for Computing Machinery, Oct. 2016, pp. 565–577. DOI: 10.1145/2976749.2978410.

[80] M Rosulek. "Improvements for Gate-Hiding Garbled Circuits". *Progress in Cryptology – INDOCRYPT 2017*. Ed. by A. Patra and N. P. Smart. Springer International Publishing, Nov. 2017, pp. 325–345.

[81] M. Naor, B. Pinkas, and R. Sumner. "Privacy Preserving Auctions and Mechanism Design". *Proceedings of the 1st ACM Conference on Electronic Commerce*. EC '99. Denver, Colorado, USA: Association for Computing Machinery, Nov. 1999, pp. 129–139. DOI: 10.1145/336992.337028.

[82] D. Malkhi et al. "Fairplay—A Secure Two-Party Computation System". *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. SSYM'04. San Diego, California, USA: USENIX Association, Aug. 2004, p. 20. DOI: 10.5555/1251375.1251395.

[83] P. Horowitz and W. Hill. *The Art of Electronics*. 3rd ed. New York, New York, USA: Cambridge University Press, 2015.

[84] D. Beaver. "Precomputing Oblivious Transfer". *Advances in Cryptology — CRYPT0' 95*. Ed. by Don Coppersmith. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, 1995, pp. 97–109. DOI: 10.1007/3-540-44750-4_8.

[85] M. Keller, E. Orsini, and P. Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer". *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria). CCS '16. New York, New York, USA: Association for Computing Machinery, Oct. 2016, pp. 830–842. DOI: 10.1145/2976749.2978357.

[86] D. Demmler, T. Schneider, and M. Zohner. "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation". *NDSS*. San Diego, California, USA: Internet Society, Feb. 2015. DOI: 10.14722/ndss.2015.23113.

[87] A. Lapets et al. *Secure multi-party computation for analytics deployed as a lightweight web application*. Tech. rep. BU-CS-TR 2016-008. Boston University, July 2016.

[88] A. Saleem et al. "Recent advancements in garbled computing: How far have we come towards achieving secure, efficient and reusable garbled circuits". *Journal of Network and Computer Applications* 108 (Apr. 2018), pp. 1–19. DOI: 10.1016/j.jnca.2018.02.006.

[89] A. Ben-David, N. Nisan, and B. Pinkas. "FairplayMP – A System for Secure Multi-Party Computation". *Proceedings of the ACM Conference on Computer and Communications Security*. Jan. 2008, pp. 257–266. DOI: 10.1145/1455770.1455804.

[90] I. Damgård et al. "Asynchronous Multiparty Computation: Theory and Implementation". *Public Key Cryptography – PKC 2009*. Ed. by S. Jarecki and G. Tsudik. Irvine, California, USA: Springer Berlin Heidelberg, Mar. 2009, pp. 160–179. DOI: 10.1007/978-3-642-00468-1_10.

[91] I. Damgård and M. Keller. "Secure Multiparty AES". *Financial Cryptography and Data Security*. Ed. by R Sion. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, 2010, pp. 367–374. DOI: 10.1007/978-3-642-14577-3_31.

[92] I. Damgård et al. "Multiparty Computation from Somewhat Homomorphic Encryption". *Advances in Cryptology – CRYPTO 2012*. Ed. by R. Safavi-Naini and R. Canetti. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[93] I. Damgård et al. "Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits". *Computer Security – ESORICS 2013*. Ed. by J. Crampton, S. Jajodia, and K. Mayes. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, 2013, pp. 1–18. DOI: 10.1007/978-3-642-40203-6_1.

[94] M. Keller et al. "Reducing Communication Channels in MPC". *Security and Cryptography for Networks*. Ed. by D. Catalano and R. De Prisco. Cham: Springer International Publishing, 2018, pp. 181–199. DOI: 10.1007/978-3-319-98113-0_10.

[95] Marcel Keller. *MP-SPDZ: A Versatile Framework for Multi-Party Computation*. Cryptology ePrint Archive, Report 2020/521. May 2020. DOI: 10.1145/3372297.3417872.

[96] I. Damgård et al. "Implementing AES via an Actively/Covertly Secure Dishonest-Majority MPC Protocol". *Security and Cryptography for Networks*. Ed. by I. Visconti and R. De Prisco. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, 2012, pp. 241–263. DOI: 10.1007/978-3-642-32928-9_14.

[97] M. Bellare et al. "Efficient Garbling from a Fixed-Key Blockcipher". *2013 IEEE Symposium on Security and Privacy*. May 2013, pp. 478–492. DOI: 10.1109/SP.2013.39.

[98] E. M. Songhori et al. "TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits". *2015 IEEE Symposium on Security and Privacy*. May 2015, pp. 411–428.

[99] Samee Zahur and David Evans. "Obliv-C: A Language for Extensible Data-Oblivious Computation". *IACR Cryptol. ePrint Arch.* (Nov. 2015), pp. 1–20.

[100] M. Hastings et al. "SoK: General Purpose Compilers for Secure Multi-Party Computation". *2019 IEEE Symposium on Security and Privacy (SP)*. May 2019, pp. 1220–1237. DOI: 10.1109/SP.2019.00028.

[101] A. D. Nicholson et al. "Characterization of gamma-ray background outside of the High Flux Isotope Reactor". *Journal of Radioanalytical Nuclear Chemistry* 318 (Aug. 2018), pp. 351–367. DOI: 10.1007/s10967-018-6097-5.

[102] Neutron Sciences at ORNL. *High Flux Isotope Reactor 2013*. [Accessed 1 June 2021] Used with permision; labels added. Aug. 2017.

[103] D. M. Hawkins. *Identification of Outliers*. Ed. by D. R. Cox. Chapman & Hall, 1980. DOI: 10.1007/978-94-015-3994-4.

[104] C. C. Aggarwal. *Outlier Analysis*. Springer, 2017. DOI: 10.1007/978-3-319-47578-3.

[105] E. M. Knorr and R. T. Ng. "Finding Intensional Knowledge of Distance-Based Outliers". *Proceedings of the 25th International Conference on Very Large Data Bases*. VLDB '99. San Francisco, California, USA: Morgan Kaufmann Publishers Inc., Sept. 1999, pp. 211–222.

[106] A. Blázquez-García et al. "A review on outlier/anomaly detection in time series data". *arXiv* (Feb. 2020).

[107] S. Basu and M. Meckesheimer. "Automatic outlier detection for time series: an application to sensor data". *Knowledge and Information Systems* 11 (Feb. 2007), pp. 137–154. DOI: 10.1007/s10115-006-0026-6.

[108] Y. Zhou et al. "A Data Quality Control Method for Seafloor Observatories: The Application of Observed Time Series Data in the East China Sea". *Sensors* 18.8 (Aug. 2018). DOI: 10.3390/s18082628.

[109] R. C. Runkle et al. "Analysis of Spectroscopic Radiation Portal Monitor Data Using Principal Components Analysis". *IEEE Transactions on Nuclear Science* 53.3 (June 2006), pp. 1418–1423. DOI: 10.1109/TNS.2006.874883.

[110] S. Minato. "Analysis of Time Variations in Natural Background Gamma Radiation Flux Density". *Journal of Nuclear Science and Technology* 17.6 (June 1980), pp. 461–469. DOI: 10.1080/18811248.1980.9732610.

[111] J. L. Burnett, I. W. Croudace, and P. E. Warwick. "Short-lived variations in the background gamma-radiation dose". *Journal of Radiological Protection* 30 (Sept. 2010), pp. 525–533. DOI: 10.1088/0952-4746/30/3/007.

[112] NIST/SEMATECH. *e-Handbook of Statistical Methods*. [Accessed 24 August 2021]. Apr. 2012. DOI: 10.18434/M32189.

[113] J. H. Ely et al. "Discrimination of Naturally Occurring Radioactive Material in Plastic Scintillator Material". *IEEE Transactions on Nuclear Science* 51.4 (Aug. 2004), pp. 1672–1676. DOI: 10.1109/NSSMIC.2003.1351968.

[114] J. H. Ely et al. "The use of energy windowing to discriminate SNM from NORM in radiation portal monitors". *Nuclear Instruments and Methods in Physics Research A* 560 (Feb. 2006), pp. 373–387. DOI: 10.1016/j.nima.2006.01.053.

[115] N. Trost and M. Iwatschenko. *Method and device for detecting artificial gamma radiation*. Patent. DE19711124C2 (Germany). Feb. 2002.

[116] D. M. Pfund et al. "Examination of Count-Starved Gamma Spectra Using the Method of Spectral Comparison Ratios". *IEEE Transactions on Nuclear Science* 54.4 (Aug. 2007), pp. 1232–1238. DOI: 10.1109/NSSMIC.2006.356110.

[117] K. K. Anderson et al. "Discriminating nuclear threats from benign sources in gamma-ray spectra using a spectral comparison ratio method". *Journal of Radioanalytical and Nuclear Chemistry* 276.3 (2008), pp. 713–718. DOI: 10.1007/s10967-008-0622-x.

[118] D. M. Pfund et al. "Low Count Anomaly Detection at Large Standoff Distances". *IEEE Transactions on Nuclear Science* 57.1 (Feb. 2010), pp. 309–316. DOI: 10.1109/TNS.2009.2035805.

[119] Pfund D. M. et al. "Improvements in the method of radiation anomaly detection by spectral comparison ratios". *Applied Radiation and Isotopes* 110 (Apr. 2016), pp. 174–182.

[120] S. Zahur, M. Rosulek, and D. Evans. "Two Halves Make a Whole". *Advances in Cryptology - EUROCRYPT 2015*. Ed. by E. Oswald and M. Fischlin. Berlin, Heidelberg, Germany: Springer Berlin Heidelberg, 2015, pp. 220–250. DOI: 10.1007/978-3-662-46803-6.

[121] Y. Ishai et al. "Extending Oblivious Transfers Efficiently". *Advances in Cryptology - CRYPTO 2003*. Vol. 2729. CRYPTO'03. Santa Barbara, California, USA: Springer-Verlag Berlin Heidelberg, Aug. 2003, pp. 145–161. DOI: 10.1007/978-3-540-45146-4_9.

[122] V. Kolesnikov and R. Kumaresan. "Improved OT Extension for Transferring Short Secrets". *Advances in Cryptology - CRYPTO 2013*. Vol. 8043. CRYPTO'13. Santa Barbara, California, USA: Springer, Berlin, Heidelberg, Aug. 2013, pp. 54–70. DOI: 10.1007/978-3-642-40084-1_4.

[123] S. Philippe et al. "A physical zero-knowledge object-comparison system for nuclear warhead verification". *Nature Communications* 7.12890 (Sept. 2016). DOI: 10.1038/ncomms12890.

Appendix A

A Comprehensive Overview of the *CypherCircuit* Package

The *CypherCircuit* package is designed to be accessible to all users, even those without extensive cryptographic expertise. The following sections provide a comprehensive introduction to the package's structure, highlighting all of its fundamental capabilities.

This introduction is adapted from the documentation provided in the *CypherCircuit* package distribution. In that original format, the code introduction is presented as a series of Jupyter notebook tutorials. A similar structure is preserved here, including code excerpts and associated outputs, to explicitly demonstrate the package's functionality.

A.1 Logic Circuits in *CypherCircuit*

This section introduces the very basics of the *CypherCircuit* API. It walks through how to set up a circuit, add wires and gates, and evaluate circuits. Once those basics have been covered, later sections will discuss how to generate truth tables and how circuits can be shared, labeled, encrypted, and decrypted.

Garbled circuits are developed using the same basic concepts as traditional logic circuits made from gates and wires. When combined in the correct order, these gates can evaluate functions based on the circuit inputs. While these functions are generally calculated discretely for binary inputs, highly sophisticated circuits can approximate continuous functions digitally.

As an example, consider a relatively simple comparator logic circuit. The comparator can take two inputs and, as the name suggests, compares the two values. A comparator circuit diagram was shown in Figure 3.1, and is reproduced below.

On the far left, the comparator accepts two input wires, w_x and w_y, each with value v_x and v_y respectively. On the far right, the comparator yields three output wires. Each input wire can take a value of either `0` or `1`, and every wire in the circuit is labeled with its wire value based on the two inputs. It can be seen that only one of these output wires will ever have a value of `1`, depending on the result of the comparison. By wire:

- **Top:** Iff $v_x < v_y$ ($v_x = 0, v_y = 1$), then $\bar{v_x} \wedge v_y = 1$

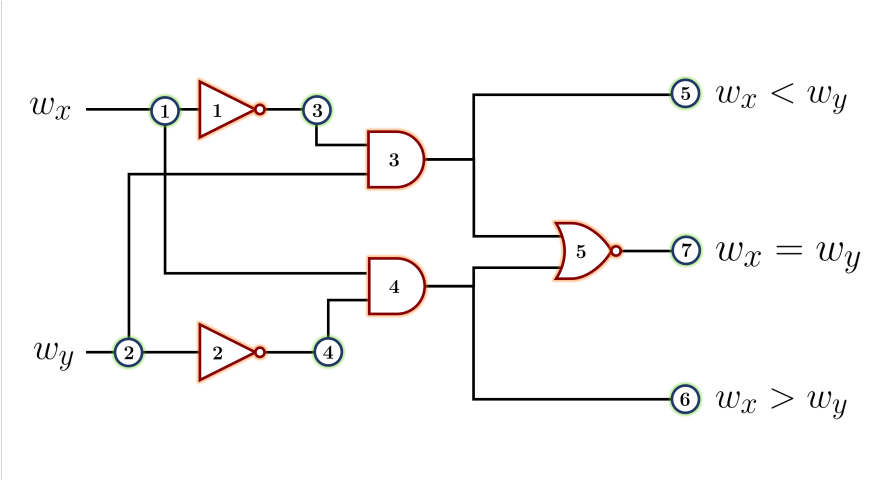


Figure 3.1

w_x	w_y	w_z
0	0	0
0	1	0
1	0	0
1	1	1

Figure 3.2a

- **Middle:** Iff $v_x = v_y$ ($v_x = 0, v_y = 0$ or $v_x = 1, v_y = 1$), then $\overline{(\bar{v_x} \wedge v_y) \vee (v_x \wedge \bar{v_y})} = 1$
- **Bottom:** Iff $v_x > v_y$ ($v_x = 1, v_y = 0$), then $v_x \wedge \bar{v_y} = 1$

For every gate in the circuit, the corresponding possible inputs and outputs can be tabulated in a truth table. An example of the truth table for an AND gate with inputs w_x and w_y was shown in Figure 3.2a.

While a single comparator can only evaluate a comparison between two bits, n comparators can evaluate a comparison over n bits. Using eight comparators chained together a user may compare any two integers from 0 to 256.

The most basic elements of the *CypherCircuit* package allow the construction of logic circuits of any size or shape, using a "build-your-own-circuit" design philosophy. The major components are outlined here, before any discussion of labeling, encrypting, or garbling the circuit.

A user begins the process by importing several types of objects from the *CypherCircuit* package:

In [1]:
```
from cyphercircuit import CircuitBoard
from cyphercircuit.wires import Wire
from cyphercircuit.gates import Not, And, Or, Nor
```

```
NEITHER THE GOVERNMENT NOR THE CONTRACTOR MAKES ANY WARRANTY, EXPRESS OR
IMPLIED, OR ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE. This notice
including this sentence must appear on any copies of this computer software.
```

A.1.1 Building Circuits: the `CircuitBoard` Object

Similar to reality, each *CypherCircuit* circuit is built upon a simulated "circuit board". Using this special `CircuitBoard` object, circuit components can be identified, tracked, operated on, and duplicated at once. Though the benefits of this construction may not be immediately obvious, the *CypherCircuit* `CircuitBoard` might be considered analogous to a canvas-like object used by a graphing package.

Programmatically, creation of a circuit board is relatively simple, with no required arguments:

```
In [2]:  circuit = CircuitBoard()
```

There are several optional arguments, but the package picks sensible defaults for anything not specified directly by the user.

Each circuit is created with a unique identifier. This identifier can be accessed using the circuit's `id` attribute. For circuits, the identifier will always begin with a `C` followed by four or more digits.

```
In [3]:  circuit.id

Out [3]:  'C0001'
```

A.1.2 Adding Wires: the `Wire` Object

Once a circuit has been created, a user may move to the next step—adding components to the circuit. The *CypherCircuit* package is designed so that circuits may be built left-to-right.

Starting with one of the simplest possible circuits—just a single AND gate—a user requires the two input wires, w_x and w_y. A wire is created using a `Wire` object, and since each wire must be "fixed" to a circuit, a wire can be passed a `CircuitBoard` object upon initialization. If a circuit is not specified, *CypherCircuit* uses a default circuit. This default circuit is generally the last circuit that was created, but this behavior can be overridden if necessary.

In this example, each wire is a part of the circuit created earlier.

```
In [4]:  w_x = Wire(circuit)
         w_y = Wire() # Defaults to ‘circuit‘, the last circuit created
```

Just like the circuit, each wire has a unique ID also accessed by the wire's `id` attribute. For wires, the ID starts with a `W` followed by four or more digits.

```
In [5]:  w_x.id, w_y.id
```

```
Out [5]:  'W0001', 'W0002'
```

Since all wires are created using the original circuit, they all contain a reference to the underlying circuit object.

```
In [6]:  underlying_circuit = w_x.circuit
         underlying_circuit.id
```

```
Out [6]:  'C0001'
```

If another circuit is created, that new circuit will have a new ID, and the wire numbering will restart.

```
In [7]:  another_circuit = CircuitBoard()
         another_wire = Wire(another_circuit)
         another_circuit.id, another_wire.id
```

```
Out [7]:  'C0002', 'W0001'
```

In a logic circuit, each of these wires carries a binary value of either `0` or `1` (or equivalently Booleans `False` or `True`). These values can be set in a couple different ways. If the wire's value is known at the time the wire is created, a user can simply pass the wire's value as an argument upon initialization. By doing this, *CypherCircuit* instruct the `Wire` object to set the `value` attribute to the given signal. Wire values must always be given as integers (in Python, Booleans are a subclass of integer, so `True` or `False` are acceptable values).[1]

```
In [8]:  wire_with_value = Wire(another_circuit, value=1)
         wire_with_value.value
```

```
Out [8]:  True
```

This `value` attribute can also be set directly.

```
In [9]:  wire_with_value.value = 0
         wire_with_value.value
```

[1]Regardless of assignment type, all wire values are stored as Booleans. Even if the wire is assigned a value as a normal integer, it will be stored (and returned) as a `True` or `False` value.

```
Out [9]:    False
```

A.1.3 Connecting Gates: the `Gate` Object

After the input wires are in place, logic gates can be introduced to the circuit. Each gate takes one or more wires as inputs and outputs a wire value that depends on both inputs and the gate type.

Logic gates are created in the circuit with one of many types of `Gate` objects. Every `Gate` object takes at least one (and usually at least two) `Wire` objects as arguments. For the case of an AND gate, this is:

```
In [10]:  and_gate = And(w_x, w_y)
```

Once again, following the same pattern as the circuit and the wire, each gate is given a unique ID. For gates, the first character is a `G`, and is followed by the customary 4 digits.

```
In [11]:  and_gate.id
```

```
Out[11]:  'G0001'
```

In this example, the AND gate's circuit was never specified. Since the AND gate is created from two input wires, it inherits its underlying circuit from those two wires.[2]

```
In [12]:  and_gate.circuit == w_x.circuit
```

```
Out[12]:  True
```

In addition to AND gates, the *CypherCircuit* package also supports NOT, OR, NAND, NOR, XOR, and XNOR gates. As expected, the specific class names for those gates are `Not`, `Or`, `Nand`, `Nor`, `Xor`, and `Xnor` respectively. For all except the NOT gate, the class constructor takes at least two input wires as arguments. (It is meaningless to have a not gate for multiple inputs and only a single output; in that case only one wire is needed.)

```
# Example gate constructors
>>>  and_gate = And(w_x, w_y)
>>>  or_gate = Or(w_x, w_y)
>>>  not_gate = Not(w_x)
```

[2]The two input wires to a gate must be part of the same circuit; violating this rule will trigger an error.

The only wire left in the simple AND gate circuit is the output wire of the gate. This wire has actually already been created for the user. Upon initialization, every `Gate` object automatically creates an output wire. This new `Wire` object may be accessed with the `output_wire` attribute of the gate, and it can be shown that the output wire has the next incremental ID.

In [13]:
```
w_z = and_gate.output_wire
w_z.id
```

Out[13]:
```
'W0003'
```

Now that the circuit includes an AND gate an the associated output wire, a user may decide to learn what the AND gate would give as an answer for a set of input values. Setting the value of input wires w_x and w_y to `0` and `1` respectively enables this calculation:

In [14]:
```
w_x.value, w_y.value = 0, 1
```

In fact, the output of every gate is automatically evaluated every time one of the gate's input wires is assigned. Using the truth table above, it is expected that the output will be zero (a `False` value).

Since w_z was defined as the output wire for the AND gate, a user can check the gate's output via the w_z wire's `value` attribute:

In [15]:
```
w_z.value
```

Out[15]:
```
False
```

If the w_x and w_y values are updated again, the circuit automatically updates the value of w_z.

In [16]:
```
w_x.value, w_y.value = 1, 1
w_z.value
```

Out[16]:
```
True
```

There are several other useful attributes that can be accessed for each gate. Along with the output wire, each gate also has a list of pointers to the input wires to the gate in the `input_wires` attribute:

In [17]:
```
[wire.id for wire in and_gate.input_wires]
```

Out[17]:
```
['W0001', 'W0002']
```

The gate type of a given gate object can be accessed using the `gtype` attribute.

In [18]:
```
and_gate.gtype
```

Out[18]:
```
'AND'
```

Additionally, the "fan-in" of the gate (the number of wires that the gate takes as inputs) can be easily accessed with the `fan_in` attribute of the gate, rather than by computing the length of the list of wires.

In [19]:
```
and_gate.fan_in
```

Out[19]:
```
2
```

A.1.4 Combining Elements to Build Arbitrary Circuits

With these basics established, more complete logic circuits can be constructed. Recall the comparator circuit shown previously in Section A.1. Below is the same comparator circuit, now with wires and gates labeled numerically: The circuit has two input wires, w_x and w_y.

In [20]:
```
comparator = CircuitBoard()
w_x, w_y = Wire(comparator), Wire(comparator)
```

Both wires pass through a NOT gate.

In [21]:
```
wire1, wire2 = w_x, w_y
gate1 = Not(wire1)
gate2 = Note(wire2)
wire3 = gate1.output_wire
wire4 = gate2.output_wire
```

The output of each NOT gate (wires 3 and 4) serves as the input to an AND gate, along with the other original wire's direct input.

In [22]:
```
gate3 = And(wire2, wire3)
gate4 = And(wire1, wire4)
```

The output of these AND gates serve as the top and bottom outputs (wires 5 and 6).

In [23]:
```
wire5 = gate3.output_wire
wire6 = gate4.output_wire
```

The middle output is then formed by passing both wires 5 and 6 through a NOR gate.

In [24]:
```
gate5 = Nor(wire5, wire6)
wire7 = gate5.output_wire
```

With the entire comparator is constructed, it is possible to test some value pairs. First, define a function to test the comparator given an input value for w_x, and input value for w_y, and an expected output. The function will throw an error if the test fails.

In [25]:
```
def test_comparator(x_value, y_value, expected_output):
    w_x.value, w_y.value = x_value, y_value
    output_values = (wire5.value, wire6.value, wire7.value)
    assert output_values == expected_output
```

If $v_x < v_y$, then only wire 5 should have a value of `True`.

In [26]:
```
test_comparator(x_value=0, y_value=1,
                expected_output=(True, False, False))
```

If $v_x > v_y$, then only wire 6 should have a value of `True`.

In [27]:
```
test_comparator(x_value=1, y_value=0,
                expected_output=(False, True, False))
```

And, finally, if $v_x = v_y$, then only wire 7 should have a value of `True`.

In [28]:
```
expected_output = (False, False, True)
# Both X and Y are 0
test_comparator(x_value=0, y_value=0,
                expected_output=expected_output)
# Both X and Y are 1
test_comparator(x_value=1, y_value=1,
                expected_output=expected_output)
```

The comparator gives exactly the expected output.

A.1.5 Circuit Diagrams for Succinct Circuit Representations

For a garbled circuit, the structure of the circuit must eventually be shared with another party. To facilitate this, a `CircuitBoard` object has a `sketch` method to generate a compact, serializable representation of the circuit.

In [29]:
```
diagram = comparator.sketch()
```

In fact, since this "diagram" takes the form of a Python dictionary, it can be converted easily into a JavaScript Object Notation (JSON) file for convenient transmission between interacting parties.

```
In [30]: diagram
```

```
Out[30]: {'class': 'diagram',
          'circuit_id': 'C0003',
          'wire_count': 7,
          'fixed_wires': {'HI': None, 'LO': None},
          'inverted_wires': {'W0003': 'W0001', 'W0004': 'W0002'},
          'freexor_wires': {},
          'input_wires': ['W0001', 'W0002'],
          'output_wires': ['W0007'],
          'gates': {'G0001': {'gtype': 'NOT',
            'input_wires': ['W0001'],
            'output_wire': 'W0003'},
           'G0002': {'gtype': 'NOT', 'input_wires': ['W0002'],
            'output_wire': 'W0004'},
           'G0003': {'gtype': 'AND',
            'input_wires': ['W0002', 'W0003'],
            'output_wire': 'W0005'},
           'G0004': {'gtype': 'AND',
            'input_wires': ['W0001', 'W0004'],
            'output_wire': 'W0006'},
           'G0005': {'gtype': 'NOR',
            'input_wires': ['W0005', 'W0006'],
            'output_wire': 'W0007'}},
          'security_parameter': 128}
```

The circuit diagram dictionary has five components that will have important values for every single circuit: the classification (e.g. that this is a dictionary of a diagram), the ID of the circuit, the number of wires in the circuit, the IDs and types of all of the gates in the circuit, and the k-bit security parameter of the circuit (to be discussed later). Additionally, there are a few other components of the diagram that are only used under specific circumstances: wires with fixed values, wires that represent the inverses of other wires, wires that benefit from the FreeXOR optimization technique, and specially designated input and output wires. Notice that the circuit diagram shares only information that meets two criteria: information that is both public and static.

- **Public**: This information is able to be shared with all parties. For instance, the structure of the circuit—equivalent to the definition of the function to be evaluated—is known by all participants.
- **Static**: This information stays constant over any number of garblings and evaluations of a garbled circuit. For example, the circuit structure never changes for any given function.

For multi-party computation, this diagram only needs to be shared once before any other computation takes place. This saves communication costs later, when encrypted tokens and input labels must be shared each time the circuit is garbled.

Also, since this package allows a circuit to be recreated from a diagram alone, no other metadata associated with the circuit needs to be included. Regenerating a circuit from the diagram is as easy as passing it as an argument upon initialization of a `CircuitBoard` object.

In [33]:
```
new_circuit = CircuitBoard(diagram)
new_circuit.id
```

Out[33]:
```
'C0003'
```

A.2 Generating Truth Tables in *CypherCircuit*

Once a circuit has been constructed, the garbled circuit protocol relies on manipulation of the truth table associated with each circuit gate. This section explains how to generate truth tables from gates in the garbled circuit using *CypherCircuit*.

Section A.1 discussed how truth tables are used to determine the outputs of a specific type of gate for all possible inputs. The gate used in that section was the AND gate, though each gate type produces a different truth table depending on the logical operation and the number of inputs that the gate accepts. For a gate with n Boolean input wires, a truth table will have 2^n rows.

The *CypherCircuit* package has several different types of tables, and as such has custom table objects for handling operations for each type. First and foremost is the `TruthTable` object. Every gate in the circuit has a `TruthTable` object which stores the truth table corresponding to that gate. Besides the `TruthTable`, the garbled circuit API also uses `LabelTable` and `TokenTable` objects. These tables provide more sophisticated labeling and encryption functionality, and receive detailed treatment in Section A.3.

Like the previous section, this section begins by ensuring that the necessary circuit components are imported.

In [1]:
```
from cyphercircuit import CircuitBoard
from cyphercircuit.wires import Wire
from cyphercircuit.gates import And, Or
```

```
Contract. You are authorized to use this computer software for Governmental
purposes but it is not to be released or distributed to the public.

NEITHER THE GOVERNMENT NOR THE CONTRACTOR MAKES ANY WARRANTY, EXPRESS OR
IMPLIED, OR ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE. This notice
including this sentence must appear on any copies of this computer software.
```

A.2.1 Generating Truth Tables: the `TruthTable` Object

For every gate created in a circuit, a `TruthTable` object is automatically generated and assigned to that gate. Each gate is presumed to be an intrinsic feature of the circuit, and so truth tables should be more or less static objects that can be generated once upon a gate's initialization. By having gate-specific truth tables, future operations including labeling and encryption of those tables can be facilitated on a gate-by-gate basis.

Before demonstrating access to a truth table, a circuit must first be constructed to include at least one gate.

In [2]:
```
circuit = CircuitBoard()
w_x, w_y = Wire(circuit), Wire(circuit)
and_gate = And(w_x, w_y)
```

Accessing the truth table for this gate is straightforward. Every gate has a reference to its corresponding truth table in the gate's `truthtable` attribute.

In [3]:
```
tt = and_gate.truthtable
tt
```

Out [3]:
```
W0001  W0002  |  W0003
----------------------
  0      0    |    0
  0      1    |    0
  1      0    |    0
  1      1    |    1
```

For easy access, `TruthTable` objects are designed to be accessed by column (or equivalently, by input wire). As an example, imagine a user who wishes to select only the first column in the AND gate's truth table. This can be accomplished using either the column's name or index. To access the column by name, use the wire's ID (`W0001`) as the truthtable's index.

In [4]:
```
and_gate.truthtable['W0001']
```

```
Out [4]:    array([False, False,  True,  True])
```

To access the column by numerical index, use the column index (0) as the truth table's index.

```
In  [5]:  and_gate.truthtable[0]
```

```
Out [5]:    array([False, False,  True,  True])
```

Another additional method exists for accessing the output column of the truth table. Rather than using either the ID (`W0003`) or index (2), the string `output` can be used instead.

```
In  [6]:  and_gate.truthtable['output']
```

```
Out [6]:    array([False, False, False,  True])
```

Similarly, the last column could also be accessed numerically using the reverse index (-1). All columns are output as Numpy arrays, since the `TruthTable` object is based on the Numpy array object.

A.3 Garbling Circuits in *CypherCircuit*

Garbled circuits are a secure method of computing a function by multiple parties: the function's arguments are jointly supplied by the involved parties, but no party exposes their inputs. The complete garbling process has three steps: labeling, encrypting, and then shuffling the circuit, all of which were described mathematically in Section 3.2.1. This section explains the labeling, encryption, and shuffling of garbled circuits using *CypherCircuit*.

To demonstrate the process in *CypherCircuit*, the AND gate with input wires w_x and w_y is used again as an example. First, a garbled circuit for this gate requires that two labels be assigned to each wire in the circuit. These labels are then used to encrypt the circuit.

This process of circuit obfuscation uses the `Wire` and `Gate` objects already discussed in Section A.1, as well as the two types of tables that were only mentioned in Section A.2 (the `LabelTable` and the `TokenTable`). Again, like in the previous sections, it is necessary to ensure that the required circuit elements are imported.

```
In  [1]:  from cyphercircuit import CircuitBoard
          from cyphercircuit.wires import Wire
          from cyphercircuit.gates import And, Or
```

A.3.1 Labeling the Circuit

Once loaded, the single AND gate can be recreated with input wires w_x and w_y, as well as output wire w_z.

In [2]:
```
circuit = CircuitBoard()
w_x, w_y = Wire(circuit), Wire(circuit)
and_gate = And(w_x, w_y)
w_z = and_gate.output_wire
```

Every wire in a circuit has a `labels` attribute. The attribute is initialized to `None` for new wires, but can be set using the wire's `set_labels` method. That method generates a dictionary of two random bit strings, one assigned to a wire value of `False` and the other assigned to a wire value of `True`. These bit strings are represented as 128-bit strings of bytes (shown below in hexadecimal notation).

In [3]:
```
w_x.set_labels(), w_y.set_labels(), w_z.set_labels()
w_x.labels
w_x.labels[True].hex()
```

Out [3]:
```
'7150fcc674ce729761fbc08f5e12404d'
```

The length of the label is required to be either 128, 192, or 256 bits in the current version of *CypherCircuit*. For generality, the security parameter will be referenced as k, but the default 128-bit security parameter will be used from now on.

In [4]:
```
circuit_128 = CircuitBoard(k=128)
w_x_128 = Wire(circuit_128)
w_x_128.set_labels()
```

Once labels have been set for all wires connecting to the gate, the values of those wires in the gate's truth table can be exchanged for the corresponding label. For a given gate, this is performed using the `label` method of the gate. That method generates a new type of table, a `LabelTable`, which behaves nearly identically to a `TruthTable`. The gate's labeled table can be accessed via the `labeltable` attribute. As a note, when displayed in tables, the k-bit byte strings are abbreviated to their hexadecimal equivalents (i.e. the binary representation of the integer 107, "1101011", converts to the hexadecimal "6b").

In [5]:
```
and_gate.label()
lt_and = and_gate.labeltable
lt_and
```

Out [5]:
```
 W0001    W0002   |   W0003
-----------------------------
62847a...  fed262...  |  c1e734...
62847a...  ed06e4...  |  c1e734...
7150fc...  fed262...  |  c1e734...
7150fc...  ed06e4...  |  d233b2...
```

For gates with many wires and inputs, manually updating labels on all wires and gates is a tedious process. Instead, each `CircuitBoard` object has the ability to update labels across the board with a single command. For example, create a new wire, w_w and connect it, along with output wire w_z, to an OR gate.

In [6]:
```
w_w = Wire(circuit)
or_gate = Or(w_w, w_z)
```

The `CircuitBoard` method `label` may be used to enact a circuit-wide label update.

In [7]:
```
circuit.label()
lt_or = or_gate.labeltable
```

To check that the AND gate has a new, relabeled table, and that the OR gate has a new labeled table of its own, check the table again.

In [8]:
```
lt_and
```

```
Out [8]:    W0001    W0002   |   W0003
            ---------------------------
            b7122...  337e52...  |  657707...
            b7122...  20aad4...  |  657707...
            8a5a4...  337e52...  |  657707...
            8a5a4...  20aad4...  |  76a381...
```

```
In  [9]:  lt_or
```

```
Out [9]:     W0003    W0004   |   W0005
            ----------------------------
            657707...  e6d1ef...  |  4cd2d4...
            657707...  f50569...  |  5f0652...
            76a381...  e6d1ef...  |  5f0652...
            76a381...  f50569...  |  5f0652...
```

A.3.2 Encrypting the Circuit

The procedure for encrypting the circuit follows the same pattern. For encryption, the input wire labels for each row in the labeled table are used to encrypt the output label. This encryption is performed by hashing together the labels for each input wire, along with the gate's ID, and then using the resulting hash digest as a one-time-pad to encrypt the output wire label.

$$E_{\ell_{w_x}^{v_x},\ell_{w_y}^{v_y}} = h\left(\ell_{w_x}^{v_x}, \ell_{w_y}^{v_y}, \langle \text{gate\_id} \rangle\right) \oplus \ell_{w_z}^{v_z} \,. \tag{A.1}$$

The hash function h is currently implemented using SHA-256.

Just like the `LabelTable`, each gate can also generate an associated `TokenTable`. Analagous to the `label` method of a gate, a `TokenTable` is generated with the gate's `encrypt` method. The encryption process for the AND gate is as follows:

```
In [10]:  and_gate.encrypt()
          tt_and = and_gate.tokentable
          tt_and
```

```
Out[10]:  ciphertexts
          -----------
            742ae2...
            006947...
            ccfe34...
            0562fc...
```

Each `TokenTable` is very similar to a `LabelTable`, except that it is one-dimensional—just storing the encrypted set of tokens. Unlike the `TruthTable` and the `LabelTable`, the one-dimensional nature of a `TokenTable` means that it is indexed by row instead of by column.

In [11]:
```
# Print the token in the ‘TokenTable‘ at index 1
token = tt_and[1]
token
```

Out[11]:
```
(b’\x00iG\x90\x00a\xe8V|\x90e\x19\xfc\xd02\x08’, 0)
```

This token includes two components: the encrypted label (in bytes) and a permutation bit. Permutation bits are discussed in the next section, so they will not be discussed in further detail here.

The encryption of the labels can be verified by trying to decrypt a row of the table. The third entry in the encrypted table should decrypt to match the third row output of the `LabelTable` when the keys are the inuts in the third row.

In [12]:
```
from cyphercircuit.encryption import decrypt_entry
```

In [13]:
```
# Get the label assigned to the third row of the truth table
assigned_label = lt_and[’output’][2]
# Get the labels assigned to the input wires
x_label, y_label = w_x.labels[True], w_y.labels[False]
# Decrypt the third row of the encrypted truth table
# (using the labels and the gate ID)
token = and_gate.tokentable[2]
decrypted_label = decrypt_entry(
    token,
    [x_label, y_label],
    and_gate.id.encode(’utf-8’),
    and_gate.circuit.security_parameter
)[0]
assert decrypted_label == assigned_label
```

This decrypted value does indeed match the third label in the AND gate’s output column.

Once again, just like in the labeling procedure, tokenization can be executed at the circuit level to avoid encrypting each gate individually. The circuit-level encryption method is simply called `encrypt`.

In [14]:
```
circuit.encrypt()
```

It can be shown that the AND gate has a refreshed table of tokens, and a new token table was created for the OR gate.

```
In [15]:  tt_and
```

```
Out[15]:  ciphertexts
          -----------
            cc215e...
            c32b2c...
            bfc485...
            270eb5...
```

```
In [16]:  tt_or = or_gate.tokentable
          tt_or
```

```
Out[16]:  ciphertexts
          -----------
            387469...
            22a3c0...
            9db37b...
            74fe6c...
```

A.3.3 Shuffling the Circuit

The last stage in the procedure is shuffling each table so that the tokens cannot be associated with the truth table values.

Unlike labeling or encryption, shuffling does not produce its own special type of table. Instead, tokens are shuffled in place within the `TokenTable`. Still, the shuffling process follows the same methodology as the labeling and encryption steps. The tokens in the `TokenTable` are shuffled with the gate's `shuffle` method.

```
In [17]:  and_gate.shuffle()
          garbled_and = and_gate.tokentable
          garbled_and
```

```
Out[17]:  ciphertexts
          -----------
            270eb5...
            bfc485...
            c32b2c...
            cc215e...
```

These tokens are the same as in the original `TokenTable`, just shuffled into a random order.

To enhance the efficiency of the garbled circuit protocol, the *CypherCircuit* package uses the point-and-permute strategy. This strategy allows the circuit evaluator to be pointed to the shuffled token that they should be decrypting, rather than trying to decrypt each token in turn.

To execute this process securely, a random permutation bit p (either a 0 or 1) is paired with each wire label ℓ matching a value of 0. (As in Chapter 3, notation $\in_R$ indicates that p_{w_x} is selected randomly from the set of $\{0, 1\}$.)

$$k^0_{w_x} = (\ell^0_{w_x}, p^0_{w_x}), \quad p_{w_x} \in_R \{0, 1\} \tag{A.2}$$

The opposite of this random bit is then paired with each wire label matching a value of 1.

$$k^1_{w_x} = (\ell^1_{w_x}, p^v_{w_x} \oplus 1) \tag{A.3}$$

Next, the encrypted table of tokens is permuted according to the permutation bits of the input wire labels. The encrypted token generated from the input wire labels with permutation bits $p_{w_x} = 0$ and $p_{w_y} = 0$ is placed first in the table. Second is the token generated from labels with bits $p_{w_x} = 0$ and $p_{w_y} = 1$, third $p_{w_x} = 1$ and $p_{w_y} = 0$, and fourth $p_{w_x} = 1$ and $p_{w_y} = 1$.

Since the bits are randomly assigned by the party generating the circuit, the other party learns nothing from the permutation bits about whether a label corresponds to 0 or 1. Also, the random assignment of permutation bits means that a table permuted according to them is also randomly shuffled. If the circuit evaluator holds any two input wire labels and matching permutation bits, they can then determine exactly which of the shuffled tokens they should decrypt, and the tokens to use to perform the decryption.

In *CypherCircuit*, each wire label and matching token are stored together in the wire's `keys` attribute.

In [18]:
```
w_x.keys, w_y.keys
```

Out[18]:
```
([<cyphercircuit.wires._Key at 0x7f8ce731e570>,
  <cyphercircuit.wires._Key at 0x7f8ce731ed90>],
 [<cyphercircuit.wires._Key at 0x7f8ce731ed50>,
  <cyphercircuit.wires._Key at 0x7f8ce731ecb0>])
```

Finally, an entire circuit can be shuffled at once, just like it can be labeled or encrypted. Following the same pattern, the circuit's `shuffle` method will systematically garble every gate in the circuit.

In [19]:
```
circuit.shuffle()
```

After running the `shuffle` method, a user can confirm that the tables of tokens belonging to the AND and OR gates have indeed been randomly shuffled.

In [20]:
```
garbled_and
```

Out [20]:
```
ciphertexts
-----------
  c32b2c...
  cc215e...
  270eb5...
  bfc485...
```

In [21]:
```
garbled_or = or_gate.tokentable
garbled_or
```

Out [21]:
```
ciphertexts
-----------
  22a3c0...
  387469...
  74fe6c...
  9db37b...
```

A.3.4 Garbling the Circuit

In most real garbled circuit implementations, the three steps of labeling, encrypting and shuffling will all be performed together at once for each gate. *CypherCircuit* enables the three to be separated; however, to avoid having to tediously write out each method, the three are also bundled together in a single `garble` method.

In [22]:
```
and_gate.garble()
```

Like each of the other methods discussed so far, the entire circuit can be garbled at once using the `garble` method.

In [23]:
```
circuit.garble()
```

Once the circuit has been generated, the user can produce garblings of the circuit which can be shared with another party. The garbling process occurs in a pipeline, with garbled chunks of the circuit being produced one at a time. Since the circuit used in this example is small, only two chunks are produced—one for the garbled tables in the circuit, and one mapping the output labels to output values. If there were many gates to be encrypted (e.g. more than 1000), there would be multiple garbling chunks with 'table' keys.

In [24]:
```
list(circuit.pipeline_garbling())
```

Out [24]:
```
[{'class': 'garbling',
  'subclass': 'table',
  'circuit_id': 'C0001',
  'number': 1,
  'tables': {'G0001': ['8f2e66c22d186c13e7f34d1b6928740900',
'e426672a18a4a3736336a227a24859f400',
'56d8784bababc0018f6ce8167b59172f01',
'11d32fc0dec92c2bea851e70a91e5f2500'],
   'G0002': ['9b7b28f5c6478de2e250ccc30b79d7c101',
'c2e793a7067aa1a8f7c77a61c34cdfc400',
'3d429553acc7fd355667da06fabe601301',
'19e3d07aff5df71cbb99f0e05f6b659500']}},
 {'class': 'garbling',
  'subclass': 'output',
  'circuit_id': 'C0001',
  'outputs': {'W0005': [0, 0]}}]
```

A.4 Encoding and Evaluating Circuits in *CypherCircuit*

This section explains how *CypherCircuit* enables a circuit to be encoded by one party, shared with a second party, and then securely evaluated by that second party. Recall that the garbling procedure created a garbled table of encrypted tokens, each corresponding to a row in a truth table. Also, notice that until this point, nothing is known *a priori* about either party's inputs. All possible inputs and outputs of the gate have been taken into account.

With the garbled circuit available, the *CypherCircuit* package allows a circuit generator to produce an encoding of a party's inputs. This is the final item required by the generator before they are able to share information with the evaluator. Three objects are shared in this process: the circuit diagram, the garbling of the tables in the circuit, and the encoding of the circuit for the generator's inputs. Additionally, the evaluator's labels are also communicated via oblivious transfer.

The process of creating a circuit diagram was discussed in Section A.1 and the garbling process was discussed in Section A.3, leaving the only the circuit encoding process left to be covered. Like in all previous sections, the process of encoding a circuit begins by importing all of the necessary *CypherCircuit* elements.

In [1]:
```
from cyphercircuit import CircuitBoard
from cyphercircuit.wires import Wire
from cyphercircuit.gates import Not, And, Nor
```

The comparator circuit introduced in the previous sections will be reused here to perform the encoding and decoding—this circuit is constructed and garbled as described in Sections A.1 and A.3.

In [2]:
```
circuit = CircuitBoard()
X, Y = wire1, wire2 = Wire(circuit), Wire(circuit)
gate1, gate2 = Not(wire1), Not(wire2)
wire3, wire4 = gate1.output_wire, gate2.output_wire
gate3, gate4 = And(wire2, wire3), And(wire1, wire4)
wire5, wire6 = gate3.output_wire, gate4.output_wire
gate5 = Nor(wire5, wire6)
wire7 = gate5.output_wire
circuit.garble()
```

A.4.1 Encoding the Circuit

Once the circuit has been created and garbled, it can be encoded with the `CircuitBoard` object's `encode` method. This method accepts a user specified vector of input values—one value for each input wire. If the party does not know the value on an input wire, they may give a value of `None`.

In [3]:
```
# The encoding party knows the value on wire 1 but not 2
vector = [0, None]
default_encoding = circuit.encode(vector)
```

By default, the `CircuitBoard` object assumes that the input wires are all wires that do not originate from some other gate and that they were created in order (specifically, ordered by increasing wire ID).[3] When a vector is encoded, each value in the vector is matched with the next wire in the predetermined set of input wires. For example, given the input vector

$$\left[v_1, v_2\right] \ , \tag{A.4}$$

[3]Wires that have "fixed" values are excluded from consideration as input wires.

values v_1 and v_2 would be assigned to circuit input wires 1 and 2 respectively.

If desired, the input wire order can be overwritten by providing circuit input wires manually. Though not strictly required, it is recommended that the circuit input wires be set when building complicated circuits.

In [4]:
```
# A redundant operation, but shown here for completeness
circuit.set_input_wires([wire1, wire2])
```

This capability extends to the output wires as well. By default, a `CircuitBoard` determines all output wires to be (in order of increasing wire ID) all those that do not terminate in a gate. For a circuit—like the comparator with output wires 5, 6, and 7—the output vector would be

$$\left[v_5, v_6, v_7\right] \ . \tag{A.5}$$

Again, this default behavior can be overwritten by explicitly declaring the output wires of the circuit. The comparator circuit actually provides a good illustration of why setting wires in this way is important. From the diagram, it is obvious that wires 5, 6, and 7 are the desired circuit outputs. However, the `CircuitBoard` will default to only registering wire 7 as an output, since both wires 5 and 6 feed into NOR gate 5. This behavior can be overridden by declaring all three wires as circuit outputs.

In [5]:
```
circuit.set_output_wires([wire5, wire6, wire7])
```

Once the circuit outputs have all be set, it remains to actually encode the circuit.

In [6]:
```
vector = [None, 1]
encoding = circuit.encode(vector)
encoding
```

Out [6]:
```
{'class': 'encoding',
 'circuit_id': 'C0001',
 'inputs': {'W0001': None, 'W0002':
   ['424d4f3834bb3f632b385ab6ac6c27e2', 1]}}
```

The method returns a dictionary with the encoded information. The first two keys contain the meta information: the class indicates that this dictionary is an encoding, and the circuit ID is used as a cursory check that the encoding matches a given circuit. The third key is a dictionary of input wires and the corresponding labels for the encoded vector. It can be shown that these two values match the labels on wires 1 and 2 for inputs 0 and 1.

In [7]:
```
print(wire1.labels[0].hex())
print(wire2.labels[1].hex())
```

```
6b8117dee75934019baa70e25dff6ff7
424d4f3834bb3f632b385ab6ac6c27e2
```

A.4.2 Sharing and Evaluating the Circuit

At this point, all of the objects that need to be communicated between parties have been created. To recap, this information has been generated in three serializable pieces:

1. A diagram outlining the general structure of the circuit (Section A.1)
2. A garbling of the circuit truth tables for randomly assigned labels and permutation bits (Section A.3)
3. An encoding of the circuit for some input vector (Section A.4)

This information is all originally assembled by the circuit generator party, and must be conveyed to the other party for evaluation. Additionally, the evaluator's inputs must be received via oblivious transfer. To do all of this communication, the *CypherCircuit* package provides a context manager through the `interaction.parties` module to execute the protocol for each party. This context manager initializes a Transmisssion Control Protocol (TCP) network connection and then handles the transfer of information between the two parties interactively.

Once this information has all been exchanged—the three objects passed from the generator to the evaluator and the evaluator's labels acquired via oblivious transfer (OT)—the evaluator may evaluate the circuit to get the answer. For every gate in the circuit with input wires w_x and w_y where the evaluator possesses the input wire labels, they may learn the corresponding output label using the decryption equation:

$$\ell^0_{w_z} = D_{\ell^1_{w_x},\ell^0_{w_y}}\left(E_{\ell^1_{w_x},\ell^0_{w_y}}\left(\ell^0_{w_z}\right)\right) \ . \tag{A.6}$$

The input wire labels could have been acquired either from the generator directly, from OT, or from the decryption of some other gate. The newly learned output wire labels are then used to decrypt subsequent gates until all output wire labels in the circuit are known.

A.4.3 The Generator and Evaluator Context Managers

To reduce the burden on a user, *CypherCircuit* provides two context managers to assist in the entire garbled circuit protocol. Both of the context managers (the `Generator` and the `Evaluator`) enable users to easily perform each step in the garbled circuit protocol. For each party, those steps are as follows:

Generator:

1. Building the circuit and sharing the diagram
2. Garbling the circuit
3. Encoding the circuit
4. Sharing the known encoded wire labels
5. Passing the evaluator's wire labels using oblivious transfer

6. Sharing the garbled tables

Evaluator:

1. Receiving the diagram and reproducing the circuit
2. Receiving the generator's encoded wire labels
3. Acquiring their own labels through oblivious transfer
4. Receiving the garbled tables
5. Decoding the circuit

To simplify the work of a user even further, each party object also has an `execute_protocol` method. This method accepts a vector with Boolean known values (and marked unknowns) for the party, and then runs through each of the protocol steps for the party.

The following example follows the role of a circuit evaluator. First, a function to simulate the circuit generator can defined and executed on a background process:

In [8]:
```
from cyphercircuit.interaction.parties import Generator
from cyphercircuit.prefab.components import (
  OneBitComparator as Comparator
)

def generator_role(address):
    """Assume the role of garbled circuit generator."""
    # Build the comparator circuit
    circuit = CircuitBoard()
    X, Y = Wire(circuit), Wire(circuit)
    Comparator(X, Y)
    # Set the last 3 wires (5, 6, and 7) to be outputs
    circuit.set_output_wires(circuit.wires[-3:])
    # Use the generator context manager
    with Generator(address, circuit) as generator:
        # The generator knows the input of wire X is 0
        # (wire Y is unknown)
        generator.execute_protocol([0, None])
```

Since the generator and evaluator communicate over the network, the a network address for communication must be specified. This address is given as an argument to the `Generator` and `Evaluator` classes upon instantiation. *CypherCircuit* can be run through a local network using the address `127.0.0.1`.

In [9]:
```
HOST = '127.0.0.1'
PORT = 65432
ADDRESS = (HOST, PORT)
```

Next, an analagous function is prepared for the evaluator:

In [10]:
```
from cyphercircuit.interaction.parties import Evaluator

def evaluator_role(address):
    """Assume the role of garbled circuit evaluator."""
    # Use the evaluator context manager
    with Evaluator(address) as evaluator:
        # The evaluator knows the input of wire Y is 1
        # (wire X is unknown)
        decoding = evaluator.execute_protocol([None, 1])
    return decoding
```

Finally, when ready, the generator's function is sent to a background process to run, and the evaluator function may be executed. Where the tutorial notebooks packaged with the *CypherCircuit* package rely on an IPython backend, the 'backgroundjobs' IPython library is shown here handling the background processes. Other networking and multiprocessing tools could be used to accomplish this instead.

In [11]:
```
from IPython.lib import backgroundjobs as bgj
jobs = bgj.BackgroundJobManager()

# Send the generator function to the background process
jobs.new('generator_role(ADDRESS)')

# Execute the evaluator's function
decoding = evaluator_role(ADDRESS)
decoding
```

Out[11]:
```
array([ True, False, False])
```

In [12]:
```
assert list(decoding) == [True, False, False]
```

It can be seen that wire 5 (the first value in the decoding) has a value of `True`, so $v_1 < v_2$, as expected. The value of wire 1 was set to be 0 by the generator and the value of wire 2 was set to be 1 by the evaluator, so the output value should indicate that wire 1 had a value less than wire 2. Neither party knew the other's input, but the circuit produced the correct output.

Of course, in a problem as small as this, knowledge of the other party's input is actually always revealed by the nature of the circuit. If $v_1 < v_2$, then $v_1 = 0$ and $v_2 = 1$. If instead $v_1 > v_2$, then $v_1 = 1$ and $v_2 = 0$. Or, if both values are equal, then each party knows that the other has the same

value. However, by extending the garbled circuit protocol to use more complicated functions and larger functions, the inputs become much less likely to be revealed by the circuit's output.

A.5 *CypherCircuit* Language Basics

This final section describing the *CypherCircuit* package demonstrates how circuits may be richly expressed using mathematical notation (rather than direct assembly using wires, gates, or other just slightly more complex components).

Rather than constructing circuits as individual wires and gates, it is much more convenient to build circuits using familiar mathematical notation. For example, in the previous sections, a comparison between numbers was performed by building a comparator circuit and feeding it the appropriate wire values. Though an example showed how the prefabricated `Comparator` component could simplify the construction step, the circuit protocol still required a user to manipulate wires and their values.

To make things easier for more general users, the `cctypes` module enables users to work with an interface that emulates traditional computer programming languages, while the package handles implementation details of the types as collections of wires and gates. The `CCInt` and `CCBool` types implement integers and Booleans as *CypherCircuit* objects.

In [1]:
```
from cyphercircuit import CircuitBoard
from cyphercircuit.interaction.parties import (
    Generator, Evaluator
)
from cyphercircuit.cctypes import CCInt, CCBool
```


As an example, a garbled circuit can be built to calculate the dot product between two vectors—one held by each party—all while preserving the privacy of the parties. Here, the two parties each

possess three integers (less than 2,147,483,648; the `cctypes` module only currently supports 32-bit integers). Together, the two parties compute and learn the dot product of their vectors.

First, the two parties agree on a circuit that performs the calculation:

$$(x_1 \cdot x_2) + (z_1 \cdot z_2) + (z_1 \cdot z_2) \; . \tag{A.7}$$

The subscript indicates which party privately holds the value.

In [2]:
```
def build_circuit():
    circuit = CircuitBoard()
    circuit.id = 'dot-product-calculator'
    # Define vector elements for two parties
    x1, y1, z1 = CCInt(), CCInt(), CCInt()
    x2, y2, z2 = CCInt(), CCInt(), CCInt()
    dot_product = (x1*x2) + (y1*y2) + (z1*z2)
    # Set the output wires
    circuit.set_output_wires(dot_product.wires)
    return circuit, (x1, y1, z1), (x2, y2, z2), dot_product
```

This `dot_product` instance of the `CCInt` type illustrates the simplification that can be achieved through this expressive syntax (the alternative method of constructing circuits would have required 192 input wires be manually fed into three 32-bit multipliers, with the outputs all piped back into two 32-bit adders).

With the circuit in hand, the generator devises a function to assign their own values to the input wires and then uses their `Generator` object to run the garbled circuit protocol. Here, they set the values on their own inputs to be 3, 4, and 5.

In [3]:
```
def generator_role(address):
    """Assume the role of garbled circuit generator"""
    circuit, (x1, y1, z1), evaluator_vector, dot_product =
   build_circuit()
    with Generator(address, circuit) as generator:
        # Set the values
        x1.set_value(3)
        y1.set_value(4)
        z1.set_value(5)
        # No need to exchange diagrams
        # (this role already has access to the circuit structure)
        generator.execute_protocol(exchange_diagram=False)
```

Next, the evaluator defines their algorithm. It is a nearly identical process; they use the same circuit description, but set the values on their own input wires x_2, y_2, and z_2 as 10, 20, and 30, respectively.

In [4]:
```
def evaluator_role(address):
    """Assume the role of garbled circuit evaluator"""
    circuit, generator_vector, (x2, y2, z2), dot_product =
    build_circuit()
    with Evaluator(address, circuit) as evaluator:
        # Set the values
        x2.set_value(10)
        y2.set_value(20)
        z2.set_value(30)
        # No need to exchange diagrams
        # (this role already has access to the circuit structure)
        decoding = evaluator.execute_protocol(
            exchange_diagram=False
        )
    return dot_product.value
```

The last thing to do before running the protocol is to define the network information required for communication between the two parties.

In [5]:
```
HOST = '127.0.0.1'
PORT = 65432
ADDRESS = (HOST, PORT)
```

Finally, the two-party protocol can be run. The expected value is 260 for the values assigned here.

In [6]:
```
from IPython.lib import backgroundjobs as bgj
jobs = bgj.BackgroundJobManager()

# Send the generator function to the background process
jobs.new('generator_role(ADDRESS)')

# Execute the evaluator's function
evaluator_role(ADDRESS)
```

Out [6]:
```
260
```