The Theory and Application of Privacy-preserving Computation

Michael R. Clark

THE THEORY AND APPLICATION OF PRIVACY-PRESERVING COMPUTATION

DISSERTATION

Michael R. Clark, B.S.C.S., M.S.C.S.

AFIT-ENG-DS-15-M-013

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-DS-15-M-013

THE THEORY AND APPLICATION OF PRIVACY-PRESERVING COMPUTATION

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Michael R. Clark, B.S.C.S., M.S.C.S.

March 2015

AFIT-ENG-DS-15-M-013

THE THEORY AND APPLICATION OF PRIVACY-PRESERVING COMPUTATION

Michael R. Clark, B.S.C.S., M.S.C.S.

AFIT-ENG-DS-15-M-013

Abstract

Privacy is a growing concern in the digital world as more information becomes digital every day. Often, the implications of how this information could be exploited for nefarious purposes are not explored until after the fact. The public is becoming more concerned about the proliferation of private data. An example of their concern comes from 2009 Dutch legislation which rejected the deployment of smart meters due to privacy concerns of the fine-grained information reporting necessary for the smart grid. Yet, there are clear benefits of the smart grid that are lost when smart metering is not available. This is true of many applications which require sensitive information to achieve their purposes. End-to-end encryption can only go so far in protecting this information. Trusted third parties could be used to assist in the processing, but they are difficult to find in large systems and represent a single point of failure.

The security community has long argued for the principle of least privilege access. In other words, access to sensitive information should only be granted if it is absolutely necessary to perform the task at hand. Interestingly, in some applications today requiring access to sensitive, personal data, it is not the actual data the involved parties need, but instead some function (e.g., sum, mean or standard deviation) of the data is needed. To follow the principle of least privilege access would be to only reveal the output of functions of the data, not the data themselves. Yet such an idea can seem paradoxical.

Solutions to this problem, referred to as privacy-preserving computation have existed since the 1980's. While initially, mostly theoretical, the solutions have been researched extensively since their original proposals, and mature enough to be used in certain practical circumstances. Yet existing protocols are still too inefficient for wide scale deployment in large systems, as described in this dissertation. This dissertation presents a new technique for privacy-preserving computation, which enables more scalable systems in a number of application scenarios.

Specifically, this dissertation proposes a new paradigm for privacy-preserving computation called transferable multiparty computation (T-MPC). Protocols for T-MPC in both the honest-but-curious (or semi-honest) and malicious adversary models are presented. These protocols are studied in two application scenarios, namely smart metering and decentralized reputation systems. In both applications, T-MPC enhances the systems. In smart metering, T-MPC enables massively scalable, in-network computations on private data. In decentralized reputation systems, T-MPC increases availability of reputation information via privacy-preserving delegation. Finally, the T-MPC protocols are compared with protocols from other privacy-preserving computational paradigms to see how their efficiencies change when switching from honest-but-curious to malicious model protocols. This is important to understand as privacy-preserving computation techniques become more widely used by industry and system designers have to decide which adversary model to operate under.

To my family for all their help in my educational endeavours.

Acknowledgements

I would like to first thank my wife and children. They are my motivation for being the best that I can be. I would also like to thank my research committee, Dr. Hopkinson, Maj. Dube, Dr. Oxley, and Maj. Laviers, who was initially on the committee but retired before I could defend. Their input through the process was invaluable. They made me think, they made me work and they made me a better researcher. Finally I would like to thank AFRL RYW and Tenet3 for their support.

Michael R. Clark

Table of Contents

Page

Page

List of Figures

List of Tables

THE THEORY AND APPLICATION OF PRIVACY-PRESERVING COMPUTATION

I. Introduction

1.1 Motivation

Whether in business, government, or the military, information can be exploited to gain a competitive advantage. Each and every day, more information is exploited to make systems more efficient, more accurate, and more reliable. Consider two simple examples. A user's or product's reputation, as determined by other users' feedback, is an often used measure when purchasing products online. This type of information has become almost ubiquitous in online purchasing. A similar feedback mechanism is also used in a number of distributed systems in a more decentralized fashion. When one node, say n_q, needs to interact with another, say n_t, n_q can query its neighbors to find out how much they trust n_t, and therefore, come up with a reputation for n_t. Another example is the smart grid. The smart grid exploits fine-grained data from consumers, suppliers, etc. to enhance the grid. A common scenario in the smart grid is to have each household's meter report their instantaneous usage back to the supplier. This information can be exploited at a supplier or distributor to better optimize generation, distribution, or to aid in purchasing or selling excess production.

The increased gathering and use of information comes with tradeoffs. Often, privacy is traded or forfeited in order to achieve the benefits described above. Privacy is one's ability to control what information is collected about them and how it is used. In reputation system, compromised privacy can lead to incentives to not be truthful when providing feedback in order to avoid retribution. The compromise of privacy can diminish the utility of the entire system. In the smart grid, the gathered information can leak other, unintended information. Researchers using this type of information have demonstrated that it can determine information such as whether or not someone is home, how many people live in a house, and what appliances are in use [1, 2]. Figure 1.1 shows a graphical example of this.

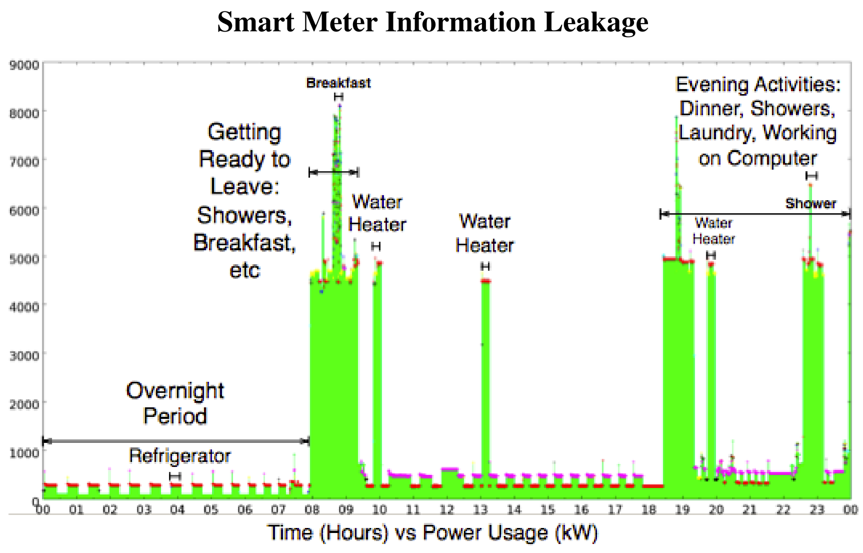


Figure 1.1: Example of private data leaked by smart meter reporting [2].

These two examples are a small sample of how privacy considerations can have real-world consequences. For better or for worse, it appears that the growth in the amount of private data being collected will continue to grow for the forseeable future. Therefore, it is imperative that users be empowered with the ability to control their privacy. There are many areas in which privacy is needed, and targeted solutions are being developed for specific domains. The most visible area is in limiting what companies are able to collect on users' online browsing. These are typically distributed as browser extensions for all the popular web browsers. They work by blocking certain code from executing within the browser, code that has been determined by experts to violate privacy in some manner. A major problem with these sorts of products is that they completely block access to private data. In the two examples above, private data is necessary for the systems to function. This dissertation deals with what to do in cases like that, where private data is necessary for a system to function.

1.1.1 Solution Techniques.

What if there were a way to still enable the sorts of benefits gained by having such sensitive information but still limit, to the maximum extent possible, the disclosure of unnecessary information? Researchers have been working on techniques to enable that since the 1980's. Broadly, this body of work is referred to as privacy-preserving computation. Privacy-preserving computation stems from the observation that, in many applications, interested parties do not need to know the inputs of the other participants, but rather they need to know a function of those inputs. In both of the example applications outlined previously, the interested party really only needs to learn statistics on the inputs (e.g., mean or standard deviation). Privacy-preserving computation allows one to better practice the principle of least privilege, that is, that a node should be given access to only the minimal amount of information necessary to do its job. In both of the example applications, this could be achieved by allowing the interested party to only learn some predefined function of the inputs (e.g., the standard deviation of power consumption in a neighborhood or the average reputation of n_t as determined by the neighbors of n_q).

Three primary classes of techniques have emerged in the privacy-preserving computation literature. The following sections describe the techniques and the scenarios where they are best used as well as review information from the literature on their related efficiencies. The contributions of this work is primarily focused on the second, secure multiparty computation.

1.1.1.1 Homomorphic Encryption.

In mathematics, a homomorphism is a structure preserving map between two algebraic structures. For example, consider the function $f : G \to H$ where G and H are groups under the operations $+$ and $\boxplus$, respectively. f is called a homomorphism if $f(x+y) = f(x) \boxplus f(y)$

for all $x, y \in G$. If f is a trapdoor function (i.e., one that is easy to compute in the forward direction, but hard to invert without special information), then f would be a homomorphic cipher. In particular, given encryptions of values, e.g. $f(x)$ and $f(y)$, homomorphic encryption allows one to compute $f(x + y)$, or an encryption of $x + y$. Such a cipher would be called partially homomorphic with respect to addition. An example of such a cipher used in practice is the Paillier cryptosystem [3]. The ElGamal cipher is an example of a multiplicatively homomorphic cipher [4].

In the late 1970's, Rivest et al. proposed the notion of a privacy homomorphism (which is the same as a homomorphic cipher) and postulated that a so-called fully homomorphic cipher could be constructed [5]. While partially homomorphic ciphers can only compute a limited set of functions on encrypted data, a fully homomorphic cipher could compute any function. Rivest et al. left the construction of such a cipher as an open problem in cryptography, which remained an open problem for another three decades. Researchers proposed several candidate constructions during those three decades, but each was broken by the cryptographic community.

Finally, in 2009, Craig Gentry proposed a fully homomorphic cipher based on ideal lattices with a sufficient security reduction to known hard problems to theoretically solve the open problem [6]. While this represented a major breakthrough in the academic cryptographic community, the practical results were still greatly lacking. In Gentry's original system, public keys were on the order of gigabytes in size, ciphertexts were also very large. Furthermore, Gentry's system required an expensive operation called bootstrapping. Since his original proposal, Gentry and other researchers have developed much more efficient systems. Currently, fully homomorphic ciphers are still slow, but could be run on powerful enough machines to evaluate complicated programs, given enough time. Gentry et al. recently implemented the AES circuit using many of these advances in the fully homomorphic encryption (FHE) literature [7]. In other words, they could evaluate

AES homomorphically, where the key, the ciphertext, and the resulting plaintext (from a decryption operation) were all privacy protected via the homomorphic cipher. They found that it took approximately 36 hours to run through the entire AES operation. Using a certain optimization which allows for SIMD (single instruction, multiple data) operations, this yields an amortized rate of around 40 minutes per block.

Considerable work remains to make FHE fast enough for real-time use. Significant advances in recent years make FHE possible in real-world applications where security and privacy trump efficiency. Partially homomorphic ciphers, however, such as Paillier and ElGamal, are quite efficient and have seen use in real-world systems.

1.1.1.2 Secure Multiparty Computation.

Around the same time Rivest was looking at privacy homomorphisms, which led to research in homomorphic encryption, Yao was exploring a related notion for distributed computation. In his seminal work, Yao proposed a problem which involves two millionaires who wish to know who is richer, yet do not want to reveal how much they are worth [8]. This formed a basis for decades of research known as secure multiparty computation (MPC or SMC). The goal of MPC is to solve the general problem of a set of parties, each with private inputs to a computation, who wish to compute some joint function of their inputs without revealing the inputs. The problem did not remain open in the theoretical sense for very long, with initial solutions coming only a few years later [9–11].

Since the initial solutions in the late 1980's, researchers in the MPC community have explored various optimizations, stronger adversary models, and better security proof techniques. Today, MPC is very fast. While Gentry et al. were exploring implementing AES using homomorphic encryption, Damgard et al. implemented AES via MPC [12]. They found they could evaluate AES in less than half a second per block. In Section 1.1.2 explores some real-world use cases involving MPC.

1.1.1.3 Functional Encryption.

In 2010, Boneh et al. formalized a new cryptographic primitive called functional encryption [13]. Functional encryption has its roots in identity-based encryption (IBE) and attribute-based encryption (ABE). In 2005, Sahai and Waters proposed the first complete IBE system [14]. IBE systems allow users to use their identity (e.g., email address or fingerprint) as their public key and retrieve the secret key associated with that identity by proving ownership of the identity to a trusted third party. The benefit of this is anyone who knows your identity can send you encrypted messages, even if you, at the moment the message was sent, do not have your secret key. ABE replaces identities with attributes. These attributes form a person's private key. A message is encrypted using a specific access policy over the attributes, and only someone holding the private key that satisfies the policy can decrypt the message. For example, in order to decrypt the message you must have the *AFIT* attribute and either the *Ph.D. Student* attribute or the *Faculty* attribute. Therefore, someone with the attribute set {*AFIT*, *Staff*} in their private key could not decrypt the message.

Functional encryption (FE) is defined as a cryptosystem in which the secret key determines what functions of the encrypted plaintext a person can learn. The holder of the secret key learns the output of the function when run on the encrypted inputs and nothing else. Both IBE and ABE can be implemented given a FE cryptosystem, but FE is clearly more powerful as it is the more generic construct. For example, your private key could give you access to the mean or standard deviation of an encrypted data set. The relationship to privacy-preserving computation is clear.

Early FE systems could only support limited functionalities (e.g., boolean formulas). Recent breakthroughs have made it possible to have FE for arbitrary functionalities [15–17]. This brings FE to the same level as FHE and MPC in terms of theoretical functionality. Implementation results of FE ciphers are not widely available in the published literature.

Naveed et al. recently published results on a variant of FE called controlled functional encryption (C-FE) and found that a single block of AES run via C-FE took approximately three minutes to run [18]. C-FE differs from FE in that it requires the user to obtain a fresh key from the authority every time it evaluates a function on a ciphertext. FE supports a number of very interesting constructs and is an important theoretical break-through. As the community gains a better understanding of the capabilities of FE, it may become a viable option for privacy-preserving computation.

1.1.2 Application Domains.

Both homomorphic encryption and secure multiparty computation have been applied to real-world privacy issues. This section describes other application areas where privacy needs have been explored and solutions have been proposed which use the building blocks described above. The purpose of this section is to illustrate the wide applicabilty of privacy-preserving computation research in general and explore application domains to which T-MPC could be applied in future work.

1.1.2.1 Electronic Voting.

Cramer et al. list three important properties of private electronic voting schemes [19].

- Verifiability: Any party should be able to verify that the final vote tally was computed correctly from all the ballots that were correctly cast.
- Privacy: An individual's vote is kept secret from any reasonably sized group of parties.
- Robust: The system can recover from faulty behavior of parties involved in the election.

Most existing systems, unfortunately, do not meet any of the three requirements above. Researchers, using techniques from privacy-preserving computation, have proposed numerous electronic voting schemes which do meet the properties listed above. Benaloh proposed a system for tally-votes (i.e., yes/no votes) which uses many of the building blocks used in today's MPC protocols, including (verifiable) secret sharing [20]. Cramer et al. proposed a scheme for similar elections using a variant of homomorphic encryption [19]. Baudron et al. proposed a scheme for multi-candidate elections (as opposed to simple yes/no votes) that uses homomorphic encryption [21]. MPC has not been directly applied in multi-candidate election systems, it is not hard to imagine how such a system could be constructed. Enumerating all of the proposed electronic voting schemes is out side the scope of the objective here, but one very mature electronic voting system worth mentioning is the Helios voting system [22]. The Helios voting system uses homomorphic encryption as well as a number of other cryptographic primitives to provide specific security properties. It has been used in a number of elections including the *International Association for Cryptologic Research*'s annual elections since 2010 and was used in campus-wide elections for student body president at Belgium's Université Catholique de Louvain. The Helios project's website reports that over 100,000 votes have been cast using Helios.

1.1.2.2 Location-based Services.

With the proliferation of mobile devices with fine-grained geolocation (typically via the global positioning satellite system), location-based services have increased in popularity [23]. In a dataset of over 310,000 apps from the Google Play Store, approximately 30% of the apps requested fine location information. Also, four out of the top ten free Android apps (accessed on 5 December 2014) in the Play Store requested fine location information.

A person's current location, or historical location information, may reveal a lot about the person. Researchers have already begun exploiting this information in order to optimize services and protocols [24–26]. Bettini et al. studied the privacy-related issues associated with location information and suggested that, given some amount of historical location information, people can be uniquely identified [27]. Additionally, researchers have developed a number of protocols to deal with the privacy issues. Relating specifically to privacy-preserving computation, Zhong et al. [28] and Solanas et al. [29] each proposed protocols which use partially homomorphic encryption to solve the privacy problem. Researchers have also explored proximity testing protocols, which allow two people to determine if they are within some distance of each other [30, 31]. All of these protocols could theoretically be instantiated with MPC techniques, though some authors have suggested that specialized protocols are faster.

1.1.2.3 Medical Information.

Vanacek [32] summed up the trend and threat in the medical community earlier this year when she wrote:

> In ten years, eighty percent of the work people do in medicine will be replaced by technology. And medicine will *not look anything* like it does today... It was estimated that 90% of all healthcare institutions will experience a data breach of some kind. Many already have. Each breach costs about $2M in fines, not to mention the loss of privacy and other incalculable costs to the patient.

Traditional information security can go a long way in the medical community in protecting private information but judicious application of privacy-preserving computation techniques can make a huge difference in limiting risk to providers and consumers. Researchers in the community have begun looking at applying these techniques to specific problems. This includes protocols that operate on DNA sequences [33–36] and protocols for classifying electrocardiogram (ECG) data for disease detection [37, 38].

These sorts of medical applications are very interesting as there are many competiting interests at play. Everyone wants the best possible diagnoses. Patients want their private data protected, but that private data can be used to make future diagnoses better. Furthermore, there are medical research companies that are developing algorithms for better detection. They want to keep their proprietary algorithms safe to maintain a competitive advantage. There are even malicious actors who would try to compromise data or algorithms for their own benefit. The examples cited above all use various privacy-preserving computation techniques, from homomorphic encryption to multiparty computation, to enable privacy while not forgoing the advances that are possible if privacy were not a concern.

1.2 Contributions

The objective addressed in this research effort is to

1. develop new protocols for privacy preserving computation and formally prove their security,
2. show how these protocols can make privacy preserving computation in the smart grid orders of magnitude more efficient than existing protocols and
3. show how the protocols can be used to enhance existing work in decentralized reputation systems, and
4. illustrate the benefits of the protocols in terms of the tradeoffs associated with adversary models.

1.3 Objectives

1.3.1 New Protocols.

Existing protocols which use homomormorphic encryption for privacy preserving computation do not scale well [39]. For example, in a smart meter network used to privately compute the sum of every meter's usage at an interval of 60 seconds, under the malicious model, networks of around 50 meters is all that can be supported. An honest-but-curious model protocol can support around 300 meters. Secure multiparty computation protocols (MPC) are more efficient but still do not scale well to the types of networks expected in practice.

Objective #1: Propose new, more flexible protocols for doing generic privacy preserving computation. Given MPC is more efficient than homomorphic encryption, these protocols will build upon existing MPC techniques. Protocols in both the honest-but-curious and the malicious adversary models are proposed. Their security is formally proven according to their respective adversary models.

1.3.2 Applications.

While privacy preserving computation techniques have existed for decades, industry is only now beginning to turn to these methods to assist in securing their systems. This is primarily because the proposed protocols are only now becoming efficient enough.

Objective #2: Demonstrate the benefits of the proposed protocols by studying two applications, smart metering and the decentralized reputation systems. For each application, system models help to contrast existing protocols with the proposed protocols to understand the benefits of T-MPC. The smart grid example illustrates scalability of the network. The decentralized reputation system example, illustrates adding new functionality to make the system more stable.

1.3.3 Adversary Model Tradeoffs.

An often overlooked aspect of deploying privacy preserving computation is the choice of adversary model. The adversary model can have a big impact on the security of the system in the real world. In the smart grid literature, a number of smart metering papers suggest the use of anti-tamper mechanisms in order to justify using a weaker adversary model. This is due to the fact that the meters are, in a sense, located in hostile environments since their physical security is not guaranteed. A similar problem could arise if a decentralized reputation system is deployed on a wireless sensor network. Little attention has been payed to the tradeoffs of different adversary models and how extra assumptions, such as anti-tamper, change these tradeoffs.

Objective #3: Develop a method for understanding the tradeoffs between adversary models.

1.4 Document Organization

Chapter 2 presents the background information needed to understand the remainder of the prospectus. Chapter 3 presents work related to **Objective #1** in developing new protocols for privacy preserving computation. Chapter 4 presents work related to application of T-MPC, which is described in **Objective #2**. Chapter 5 presents work related to adversary modeling described in **Objective #3**.

II. Preliminaries and Related Work

This chapter, presents the background information necessary to understand the remainder of the dissertation. Furthermore, the information presented in this chapter serves as a discussion of related works upon which this research builds. This will help the reader to understand the contributions to the research area.

2.1 Adversary Modeling

There are two primary adversary models seen throughout the privacy preserving computation literature, the honest-but-curious (HbC) model, which is sometimes referred to as semi-honest, and the malicious model [40]. In the HbC model, adversaries follow the protocol exactly as specified. For example, an HbC adversary in the smart grid application would always use the correct input. The corrupt parties do, however, collude by using information gathered during the execution of the protocol to attempt to violate an honest party's privacy.

The malicious adversary model represents the other end of the spectrum in privacy preserving computation. Malicious adversaries will deviate from the protocol in any way to attempt to violate another's privacy. An example of this would be using non-random values when the protocol specifies random numbers or having all corrupt parties collaborate to choose their random values. This model is much stronger than the HbC model because it makes fewer assumptions about adversary behavior. That comes at a cost, usually in efficiency and more complex cryptographic protocols.

Another common assumption related to adversary models seen throughout the literature is an assumption on the number of corrupt parties in a given protocol execution. Many protocols that use homomorphic encryption are able to achieve full threshold security. In other words, such protocols can guarantee security as long as there is at least one non-corrupt party. Multiparty computation (MPC) protocols have traditionally required lower thresholds. For example, with HbC an often made assumption is that fewer than half of the parties are corrupt. In the malicious model, the assumption is often that less than a third are corrupt. More recent MPC protocols are able to achieve full threshold security but at the expense of requiring computationally expensive operations (e.g., zero-knowledge proofs).

2.2 Homomorphic Encryption with the Paillier Cryptosystem

Homomorphic encryption is a special encryption paradigm which can enable privacy preserving computation. It allows an untrusted party to perform operations on ciphertexts such that it has a well-defined effect on plaintexts. For example, given the encryptions of two values, say $\mathcal{E}(x_1), \mathcal{E}(x_2)$, computing some operation on the ciphertexts, say $\boxplus$, results in addition of the plaintexts. Mathematically, this is $\mathcal{E}(x_1) \boxplus \mathcal{E}(x_2) = \mathcal{E}(x_1 + x_2)$. Decrypting the results reveals $x_1 + x_2$. Note that this requires no knowledge of x_1 or x_2. Fully homomorphic encryption is a cipher that supports addition and multiplication and allows us to compute any function. Existing FHE ciphers (e.g., Gentry's original work [6] and the BGV cipher [41]) are still not efficient enough to be considered for practical applications involving large numbers of parties.

The primary cipher used in this work is the Paillier cipher [3]. This cipher is additively homomorphic, meaning it supports addition operations on the plaintext values using only ciphertexts. Below is a description of a simplified version of the Paillier cipher and a description of how to perform the additive homomorphic operation. A public key cipher consists of three routines: **KeyGen** which generates a public and a private key, **Enc** which returns a ciphertext given the public key and a message, and **Dec** which returns a message given a ciphertext and the private key. The mathematics of each routine is given below.

Routine **KeyGen**

1. Let p, q be large primes of the same size.
2. $n \leftarrow pq$, $\lambda \leftarrow \varphi(n) = (p-1)(q-1)$
3. $g \leftarrow n+1$, $\mu \leftarrow \lambda^{-1} \bmod n$
4. Encryption (public) key $K_e = (n, g)$, Decryption (private) key $K_d = (\lambda, \mu)$

Routine **Enc**

Given: Plaintext m and $K_e = (n, g)$

1. Choose at random $r \in \mathbb{Z}_n^*$
2. Ciphertext $c \leftarrow g^m \cdot r^n \bmod n^2$

Routine **Dec**

Given: Ciphertext $c \in \mathbb{Z}_{n^2}^*$ and $K_d = (\lambda, \mu)$

1. Message $m \leftarrow L(c^\lambda \bmod n^2) \cdot \mu \bmod n$

Where $L(u)$ returns the quotient of $(u-1)/n$.

2.3 Secure Multiparty Computation Protocols

Multiparty computation (MPC) was first introduced in the 80's with solutions to the problem falling into two main classes, 1) garbled circuits [9] and 2) secret sharing, both in the computational setting [10] and in the information theoretic setting [11]. MPC deals with the problem of having parties $p_1, \ldots, p_n$ with inputs $x_1, \ldots, x_n$ who wish to compute some function of their inputs, say $f(x_1, \ldots, x_n)$, without revealing their individual inputs. The output(s) of the function is revealed to potentially any group of the parties or even authorized third parties. The focus of this work is on the secret sharing variant of MPC, and in particular, the information theoretic setting with honest majority. These protocols are simpler, but the general general ideas behind T-MPC can easily be extended to other settings (e.g., dishonest majority with computational security).

Below is an overview of MPC that outlines a generic MPC protocol. For simplicity, some of the details are abstracted away as they are not necessary for the purposes of this dissertation. Furthermore, malicious model MPC protocols are often divided up into two phases, preprocessing and computation. Doing this allows the protocols to be proven secure even in the face of malicious adversaries. For the purposes of this work, the description below focuses on the computation phase.

Given a finite field, $\mathbb{F}$, and each $x_i \in \mathbb{F}$ for $i = 1, \ldots, n$. Let $(s_1, \ldots, s_n) \leftarrow$ **share**$(x_i, seed)$ be a secret sharing function, which takes the secret to be shared as its first input and a random seed as its second input. The random seed is used to generate the random values needed by the specific secret sharing function. The outputs, $s_1, \ldots, s_n \in \mathbb{F}$, form the n shares of the secret. Each party p_i uses **share** to generate n shares of their input x_i. Associated with the function **share** is a threshold t which determines the number of shares needed to reconstruct the input. Each other party receives one of the shares over a secure channel. The most common choices for implementing **share** in the MPC literature are Shamir secret sharing [42] and additive secret sharing. Additive secret sharing relaxes **share** by requiring a cyclic group, instead of a finite field.

After all inputs are shared among the parties, they represent the function f as an arithmetic circuit with addition and multiplication gates. They use functions **add** and **mult** to evaluate addition and multiplication gates of the arithmetic circuit. For example, say the parties hold shares of a and b (called a_i and b_i) and want to compute $c = a + b$ (resp., $c = a \cdot b$), they each call $c_i =$ **add**(a_i, b_i) (resp., $c_i =$ **mult**(a_i, b_i)) to compute a share c_i of c. Full details of the operation of these functions can be found in [10, 11, 43], for example, for details on how the functions **add** and **mult** can be constructed both in the honest-but-curious and malicious models.

Embedded within the circuit description are the identities of the parties authorized to learn the output(s) of the function. At the end of the computation of the circuit, the parties send the appropriate shares of the output(s) to the party or parties authorized to learn a given output. The authorized parties run the function **recombine** which takes n shares of the output value (only $t + 1$ shares are necessary, but the adversary models associated with MPC assumes all n are available) and returns the output value. Many malicious model MPC protocols also include a preprocessing phase where the parties generate helper data to increase security of the computation of **share**, **recombine**, **add**, or **mult**. The helper data should not depend on the inputs or the function to be computed (beyond minimal information such as the number of multiplication gates). Preprocessing is discussed in more detail in Chapter 4.

2.4 Secret Share Redistribution

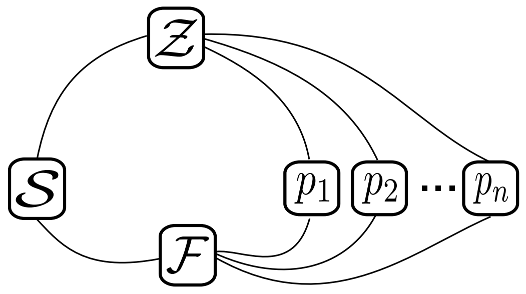
The main idea of secret share redistribution is that a certain number of parties receive shares of a secret. Say they wish to redistribute their shares to a new (potentially overlapping) set of parties that may have a different reconstruction threshold without reconstructing the original secret. Secret share redistribution protocols aim at solving this problem.

The simplest redistribution protocol is very similar to the multiplication protocol already used by MPC and was proposed by Desmedt et al. [44] It is secure in the honest-but-curious model and works by having each party share their share with the new parties (i.e., they create subshares). This will work for any linear secret sharing method such as Shamir or additive secret sharing. A number of malicious model redistribution protocols have been proposed (e.g., [45] and subsequent works). Unfortunately, these will not work for for malicious model T-MPC as they do not provide semantic security. Therefore Chapter 3 presents a new malicious model redistribution protocol.

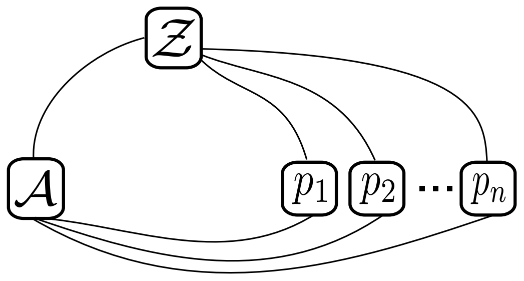
2.5 Universally Composable Security Framework

Proving the security of protocols can be a difficult process. One technique that has been used in many areas of cryptography, and in particular, in multiparty computation, is the universally composable security (UC) framework due to Ran Canetti [46]. This section reviews basics of the framework and describes how to prove security in it. For further details the UC Framework, see Canetti's work cited above and Sections 1.3 and 3.4 of [47]. The description here focuses primarily on privacy concerns in multiparty computation.

The basic setup of the UC framework begins with two worlds, the ideal world and the real world, and a number of parties $p_1, \ldots, p_n$. In the ideal world, an ideal functionality $\mathcal{F}$ is used that is secure by definition (often using a trusted third party). In the real world, a protocol π which realizes the ideal functionality is run. The goal is to show that π is as secure as $\mathcal{F}$. In each world there is an entity called the environment, denoted $\mathcal{Z}$. In each world, there is an additional party, the adversary $\mathcal{A}$ in the real world and a simulator $\mathcal{S}$ in the ideal world. These parties communicate their views of their respective worlds back to $\mathcal{Z}$. To keep the setup as generic as possible, let $\mathcal{Z}$ provide the inputs to the parties. This setup is shown in Figure 2.1.



(a) Ideal World.



(b) Real World.

Figure 2.1: Basic universally composable security framework setup.

Cannetti proves two fundamental theorems behind the UC security framework. The UC security theorem says that if $\mathcal{Z}$ cannot distinguish between the two worlds, then the protocol π, run in the real world, is as secure as the ideal functionalty (which was secure by definition). Below is a description of how to do this, focusing on privacy concerns. Note, however, that the UC framework does not only apply to privacy. The description below is adapted from the description of Cramer et al. in [48].

In the real world, the execution of π leaks information to $\mathcal{A}$, who in turn, shares his view of the world with $\mathcal{Z}$. Let $\mathcal{L}_\pi$ be the information leaked during execution of π, which forms $\mathcal{A}$'s view of the world. In the ideal world, let $\mathcal{F}$ leak only very specific information to $\mathcal{S}$, information that can readily be shown to not violate any party's privacy. Call this information $\mathcal{L}_\mathcal{F}$. The job of $\mathcal{S}$ is to use $\mathcal{L}_\mathcal{F}$ to generate $\mathcal{L}_\pi$. If $\mathcal{S}$ can do this, since $\mathcal{L}_\pi$ is generated from $\mathcal{L}_\mathcal{F}$, it contains no more information (from an information theoretic prospective) than $\mathcal{L}_\mathcal{F}$. Since $\mathcal{L}_\mathcal{F}$, by definition, does not violate anyone's privacy, $\mathcal{L}_\pi$ also does not violate anyone's privacy. Therefore, π does not violate anyone's privacy. Note that in practice, no two runs of π will result in identical $\mathcal{L}_\pi$ sets. Similarly, the $\mathcal{L}_\pi$ generated by $\mathcal{S}$ will not be identical to a specific $\mathcal{L}_\pi$ from the real world. Therefore, instead of requiring equivalence, it must be shown that the $\mathcal{L}_\pi$ generated by $\mathcal{S}$ be indistinguishible from the leaked information sets coming from the real world.

The second theorem of Canetti's work, the composability theorem, states that UC-secure protocols can be composed with each other and themselves in arbitrary, even adversary controlled, ways and the resulting composition remains secure. The result of this second theorem is a complex protocol can be decomposed into many smaller, subprotocols. The security of the the subprotocols is proven using the first theorem. Finally, the composition is proven secure using the second theorem. The security proofs of the T-MPC constructions in Chapter 3 utilize both of the theorems described above.

III. Transferable Multiparty Computation

Existing methods for privacy-preserving computation using both homomorphic encryption and secure multiparty computation (MPC) have issues that are described in Chapter 4, when applied to the applications studied in this dissertation. This chapter proposes a new paradigm for privacy preserving computation that builds upon MPC to enable both computations that are more efficient and advanced functionality not previously available. The protocols are not tied to specific applications, however, and are more generally useful. The key observation behind the protocols is that prior work has assumed a static set of computation parties. The protocols presented here remove this limitation, which creates numerous benefits. The paradigm is called transferable multiparty computation (T-MPC), and this chapter gives an overview of T-MPC as well as protocol constructions and security proofs in both the honest-but-curious model and the malicious model.

3.1 Overview

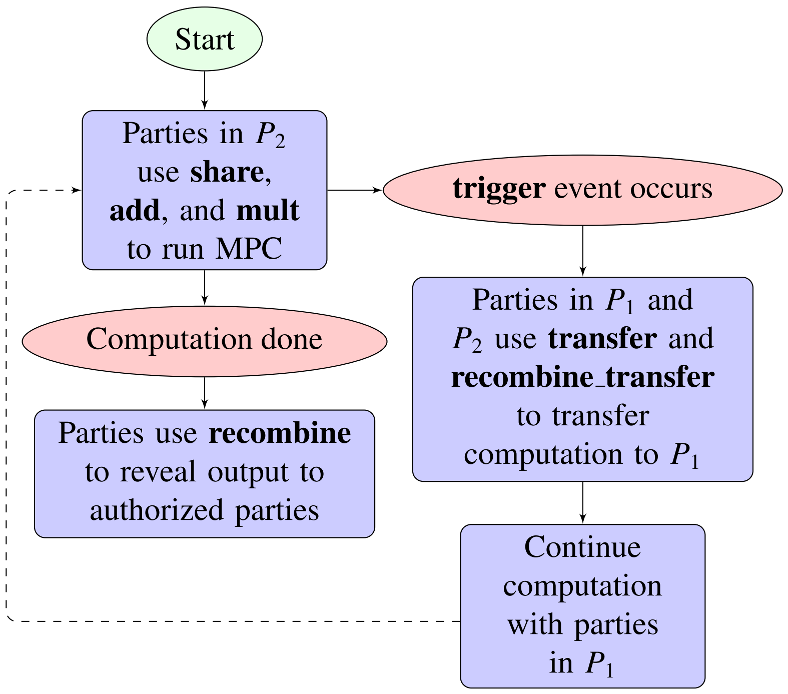
MPC allows a static set of parties to compute some function of their private inputs without revealing the inputs. T-MPC relaxes the restriction of having a static set of parties. While this relaxation might seem simplistic on the surface, constructing T-MPC protocols is non-trivial and offers numerous benefits, which are described in Chapter 4. For simplicity, the description below depicts T-MPC as having two sets of parties P_1 and P_2 (which may or may not overlap and can have different sizes). T-MPC can support any number of sets of parties.

Let $|P_1| = n'$ and $|P_2| = n$. The protocol descriptions also assume that one set, namely P_2, is performing a computation and transfers that computation to the other set, namely P_1. This is solely for simplicity as T-MPC allows for more generic constructions.

Any number of sets can be performing multiparty computations (possibly in parallel) and can transfer their computation to any other set (or possibly many other sets). Therefore, while the present description of T-MPC is out of necessity simplified, the application of T-MPC can vary greatly. In the honest-but-curious case assume that less than $\lceil n/2 \rceil$ ($\lceil n'/2 \rceil$, respectively) parties are corrupt as this is the theoretical maximum for information theoretic MPC protocols. For the malicious case the theoretic maximum is $\lceil n/3 \rceil$ ($\lceil n'/3 \rceil$, respectively).

T-MPC builds upon the description of MPC from Section 2.3. T-MPC begins with the functions **share**, **recombine**, **add**, and **mult**. T-MPC adds a signaling mechanism called **trigger**. This mechanism interrupts an ongoing multiparty computation running among parties of P_2 and forces a transfer of the computation to the parties of P_1. The **trigger** mechanism can be configured to run, for example, periodically, upon certain events (e.g., failure of a party), inserted manually into the arithmetic circuit, etc. Chapter 4 describes two specific trigger functions for two applications in more detail.

T-MPC adds two additional functions, **transfer** and **recombine_transfer**. **transfer** takes the share of the input to be transferred as the first input and a random seed as the second input. It outputs n' subshares of the input. **recombine_transfer** takes n' subshares as its input and outputs a new share to be used in private computations. These two functions exploit a redistribution property of linear secret sharing schemes which allows an authorized subset of parties holding shares of a secret value, say s, to redistribute s to a new set of parties without revealing s [44]. In particular, for T-MPC, the parties in P_2 run **transfer** to generate subshares for parties in P_1 of every value needed for P_1 to continue the multiparty computation. Parties in P_1 use **recombine_transfer** to compute new shares of these values. T-MPC is constructed using all of the functions specified above as shown in Figure 3.1. Section 3.2 presents details for these functions under the HbC model, and Section 3.3 presents the detailed functions in the malicious model.


Start
Parties in P_2 use **share**, **add**, and **mult** to run MPC
trigger event occurs
Computation done
Parties use **recombine** to reveal output to authorized parties
Parties in P_1 and P_2 use **transfer** and **recombine_transfer** to transfer computation to P_1
Continue computation with parties in P_1
Figure 3.1: T-MPC state diagram.

3.2 Honest-but-Curious Realization

The basic functions of MPC (i.e., **share**, **recombine**, **add** and **mult**) used by HbC T-MPC are due to Ben-Or et al. [11]. **share** and **recombine** are both handled using Shamir secret sharing. Briefly, to generate shares s_i of s such that any subgroup of $t+1$ out of the n parties can reconstruct s, construct the polynomial $\sigma(x) = s + r_1 x + r_2 x^2 + \cdots + r_t x^t$ in the finite field where the coefficients r_j are chosen randomly from the field. The share $s_i = \sigma(i)$ is sent to party i. Any subgroup of at least $t+1$ parties can reconstruct s by exchanging their shares with one another and using Lagrangian interpolation to compute $\sigma(0) = s$. To compute **add** on two secret inputs, say s and s', each party simply adds their shares of the inputs. No communication is required at all because of the linearity of the secret sharing method. To compute **mult** on two secret inputs s and s', each party p_i computes $m'_i = s_i \cdot s'_i$ then uses Shamir secret sharing to create subshares of m'_i, distributes a subshare to each of the other parties. Party p_i then uses Lagrangian interpolation on the subshares they receive from the other parties to compute m_i which turns out to be a valid sharing of $m = s \cdot s'$. These functions are combined together into a single honest-but-curious MPC protocol π_{mpc_h}.

The new functions, **transfer** and **recombine_transfer**, T-MPC are fairly simple in the honest-but-curious model. Together these form the honest-but-curious transfer protocol π_{T_h}. Assume that upon a **trigger** event, parties in P_2 must transfer the input value s to parties in P_1. The redistribution function needed for **transfer** and **recombine_transfer** works almost exactly like the multiplication protocol just described. Each party $p_j \in P_2$ with share s_j of s, computes n' subshares of s_j using Shamir secret sharing with threshold t' and sends subshare s_{ji} to party $p_i \in P_1$. Party p_i, after collecting all subshares, computes **recombine_transfer** by running Lagrangian interpolation on the subshares to compute the new share s_i of s. This process is described in Figure 3.2.

Protocol π_{T_h}

Given: $P_1, P_2 \subseteq P$ where each $p_j \in P_2$ holds a share s_j of s, and $|P_1| = n'$, $|P_2| = n$ and $t' = \mathbf{I}(n'/2)$ and $t = \mathbf{I}(n/2)$ respectively (where $\mathbf{I}$ is the integer just larger than the parameter).

transfer: Upon a trigger event

1. Each p_j generates subshares $s_{1j}, \ldots, s_{n_i j}$ of s_j using Shamir secret sharing with a threshold of t' and a random polynomial with coefficients $(s_j, r_{j1}, r_{j2}, \ldots, r_{jt'-1})$.
2. Each p_j sends the subshare s_{ij} to party $p_i \in P_1$ over the private channel.

recombine_transfer

1. Each $p_i \in P_1$ receives n subshares s_{ij}.
2. p_i then uses Lagrangian Interpolation to recover their new share s_i

Figure 3.2: Honest-but-curious **transfer** and **recombine_transfer** functions.

3.2.1 Correctness and Complexity.

Security and correctness proofs for the transfer protocol for redistributing secret shares from P_j to P_i follow easily from [44]. That work guarantees that, first, subsets of participants in P_i up to size t_i have no information about the secret. Second, as long as at least t_j parties of P_j erase their information about s (both their original shares and the subshares they created), then s can only be recovered by parties in P_i. This is guranteed due to the assumed threshold of corrupt parties. Finally, subsets of parties in P_i and P_j cannot collude to gain extra information about s as long as their sizes are no bigger than t_i and t_j respectively.

Since generating shares and recombining shares using interpolation can be done in parallel by all parties, these methods are quite efficient. Straightforward algorithms for generating the shares (polynomial evaluation) and recombining (Lagrange interpolation) are quadratic, but there exist $O(n \log^2 n)$ algorithms for each [49]. Even the quadratic algorithms would be sufficiently fast for most applications. As the n_j parties must communicate each of their n_i shares, the communication complexity is $O(n_j n_i)$ (assuming only one communication channel). Since really only the threshold t needs to be satisfied, the computation and communication requirements can be made even more efficient in practice.

3.2.2 Security.

It seems to make sense that the T-MPC protocol suggested above is secure as long as the chosen MPC protocol is secure. This would seemingly follow from the fact that the transfer protocol itself is secure. A more rigorous approach to proving the security of the proposed protocol is needed. This is achieved by using the Universal Composability (UC) framework.

The UC framework provides an elegant approach for proving the security of the T-MPC protocol as there already exist UC-secure MPC protocols. Therefore, proving security of the T-MPC protocol requires proving the UC-security of the transfer protocol as the T-MPC protocol is the composition of an MPC protocol and the transfer protocol. Since two UC-secure protocols can be composed in arbitrary ways and still be UC-secure, the security of the T-MPC protocol follows directly from the composition theorem. Following the UC framework, descriptions for the ideal functionality and details on how $\mathcal{Z}$ cannot distinguish between the two worlds is presented below.

Ideal Functionality:

1. Each party in P_j sends its share of s (i.e., s_j) to $\mathcal{F}$.
2. $\mathcal{F}$ uses the shares to reconstruct s and generates new shares of s, say s_i', with threshold $n_i/2$.
3. For each corrupt party $p_j \in P_j$, $\mathcal{F}$ sends s_j to $\mathcal{S}$.
4. For each corrupt party $p_i \in P_i$, $\mathcal{F}$ sends s_i' to $\mathcal{S}$.
5. $\mathcal{F}$ sends the share s_i' to $p_i \in P_i$.

Security Proof: The ideal functionality just described is very simple and clearly secure by definition as $\mathcal{F}$ acts as a trusted third party. Furthermore, due to the assumed threshold of corrupt parties, the shares that $\mathcal{F}$ leaks to $\mathcal{S}$ do not violate anyone's privacy. These shares form the set $\mathcal{L}_{\mathcal{F}}$. To prove security of the real world protocol, the $\mathcal{S}$ and $\mathcal{F}$ interact to make the ideal world indistinguishable from the real world.

Theorem 1. *The environment $\mathcal{Z}$ cannot distingiush between the real world and the ideal world.*

Proof. In the real world, $\mathcal{Z}$'s view consists of information learned by $\mathcal{A}$. This includes: 1) shares held by corrupt parties in P_j, 2) new shares held by corrupt parties in P_i, 3) subshares generated by corrupt parties in P_j, and 4) subshares sent to corrupt parties in P_i. These four pieces form the set $\mathcal{L}_\pi$. In other words, let C_j be the set of corrupt parties in P_j and C_i be the set of corrupt parties in P_i, then $\mathcal{L}_\pi = \{\{s_j : j \in C_j\}, \{s'_i : i \in C_i\}, \{s_{ji} : i \in 1, \ldots, n', j \in C_j\}, \{s_{ji} : i \in C_i, j \in 1, \ldots, n\}\}$.

Notice that $\mathcal{L}_\mathcal{F}$ is basically the first two items. In other words, $\mathcal{L}_\mathcal{F} = \{\{s_j : j \in C_j\}, \{s'_i : i \in C_i\}\}$. So, $\mathcal{S}$ must compute the third and fourth items using only what is in $\mathcal{L}_\mathcal{F}$ and do so in a manner which makes the resulting set indistinguishable from $\mathcal{L}_\pi$ in the real world. Using the shares of corrupt parties in P_j, $\mathcal{S}$ can simply create subshares of these to form number 3 from above.

Number 4 is a litte more difficult to simulate as each corrupt party in P_i will receive a subshare from each corrupt party in P_j. In other words, some of the subshares that $\mathcal{S}$ just generated to satisfy number 3 have to be used directly to satisfy number 4. For the remaining subshares needed to satisfy number 4, simple random values cannot be used as the interpolated polynomial would likely not have the correct degree. The degree of the polynomial, t_i, is $\lfloor n_i/2 \rfloor$. Let c be the number of corrupt parties in P_j. Therefore, for each corrupt party $p_i \in P_i$, the simulator already has c shares to satisfy number 4. $\mathcal{S}$ sets a zeroth share to be the new share for p_i that comes from number 2 above. It then picks $n_i/2 - c$ other shares at random. Using these shares, Lagrangian interpolation returns the sharing polynomial σ. If σ has degree t_i, the simulator's job is done. The other option is that σ has degree less than t_i, this is not likely to occur due to the randomly chosen shares, but if it does, the process is repeated with new random shares. Once a σ with degree t_i is found, $\sim$, generates the remaining shares for each corrupt party in P_i to satisfy number 4.

If the set $\mathcal{L}_\pi$ that the simulator just generated is statistically indistinguishable from the $\mathcal{L}_\pi$ that results from executing π from the point of view of $\mathcal{Z}$, then the protocol is UC-secure. There were basically four parts to these sets described above. Indistinguishability of the first two comes directly from the fact that the adversary does not control enough parties to run Shamir's reconstruction step. Therefore, they each contain no information in the information theoretic sense. Indistinguishability of the third comes from the fact that in both worlds, these subshares are valid subshares. Like generation, showing indistinguishability of the fourth is a little more difficult. In the real world, the subshares in number 4 result in valid new shares held by corrupt parties in P_i. Recall though, that since the adversary does not control enough parties in P_i that the new shares are random from the adversary's point of view. Given the way the subshares in number 4 are constructed, they form valid subshares of the new shares found in number 2 and come from a correct degree polynomial. Furthermore, the subshares are random. Therefore, they are also indistinguishable in the two worlds. So, the protocol in Figure 3.2 is UC-secure. $\square$

3.3 Malicious Model Realization

This section presents the malicious model realizaation of T-MPC which builds upon the HbC realization presented in Section 3.2. Similar to before, the malicious model realization uses an MPC protocol secure in the malicious model. The malicious model MPC protocol, built with malicious model variants of **share**, **recombine**, **add**, and **mult** used in the T-MPC protocol below comes from [50]. This section focuses on malicious model variants of **transfer** and **recombine_transfer** and constructs a single protocol π_{T_m} from these two.

The malicious model transfer protocol works as follows. For **transfer**, each party in P_2 shares their share of s with the parties in P_1. **recombine_transfer** exploits an observation due to McEliece and Sarwate. Since fewer than a third of the parties are corrupt, Shamir secret sharing reconstruction can be made robust using Reed-Solomon decoding. This is because Shamir secret sharing is just a special instance of Reed-Solomon coding [51]. Under normal circumstances robustness is not enough for malicious model MPC as it assumes that the dealer is honest, but in T-MPC, the shares that a party in P_1 receives come from each party in P_2. Therefore, there is no one single dealer. Since there are enough honest parties in P_2, each $p_i \in P_1$ receives enough honest shares to be guaranteed to recover the correct new share. Figure 3.3 shows the details of the protocols.

Protocol π_{T_m}

Given: $P_1, P_2 \subseteq P$ where each $p_j \in P_2$ holds a share s_j of s, and $|P_1| = n'$, $|P_2| = n$ and $t' = \lfloor n'/2 \rfloor$ and $t = \lfloor n/2 \rfloor$ respectively.

transfer: Upon a trigger event

1. Each p_j generates subshares $s_{1j}, \ldots, s_{n_i j}$ of s_j using Shamir secret sharing with a random polynomial with coefficients $(s_j, r_{j1}, r_{j2}, \ldots, r_{jt'-1})$.
2. Each p_j sends the subshare s_{ij} to party $p_i \in P_1$ over the private channel.

recombine_transfer

1. Each $p_i \in P_1$ receives n subshares s_{ij}.
2. p_i uses a Reed-Solomon decoder on the subshares to recover s_i.

For a small number of parties, a simple bruteforce RS decoder is faster than more sophisticated decoders. For this decoder each subgroup of shares of size t is run through the regular Lagrangian interpolation algorithm to get a putative secret. The putative secret that occurs most often is chosen as the secret. Due to the number of corrupt parties and the threshold used for sharing this will be the correct secret with overwhelming probability [51].

Figure 3.3: Malicious **transfer** and **recombine_transfer** functions.

3.3.1 Complexity.

The complexity of the malicious model T-MPC functions **transfer** and **recombine_transfer** changes in comparison with the HbC functions due to the fact that a Reed-Solomon decoder is used to ensure robustness. Fast $O(n \log^2 n)$ as well as straight forward $O(n^2)$ decoders exist [51]. For small sets of parties, a brute-force, $O(n!)$ decoder is faster. The brute-force decoder is described in Figure 3.3.

3.3.2 Security.

Security of the malicious model T-MPC functions is proven using the same paradigm that was used in the HbC case. The setup in this case, however, is more complex because corrupt parties can deviate from the protocol specification. A common technique for handling this is to let the simulator $\mathcal{S}$ run a (black box) copy of the adversary's code $\mathcal{A}$. The ideal functionality is shown below. Following the ideal functionality, a high-level overview of the security proof is given. The details of the proof follow directly from the techniques used in the HbC proof.

Ideal Functionality:

1. The honest parties in P_j send their shares to $\mathcal{F}$.
2. $\mathcal{F}$ uses the shares to generate the shares of corrupt parties in P_j and sends these generated shares to $\mathcal{S}$.
3. $\mathcal{F}$ creates subshares for each of the honest parties' shares and sends any of these which would be delivered to corrupt parties in P_i to $\mathcal{S}$.
4. $\mathcal{F}$ receives subshares for the corrupt parties in P_j from $\mathcal{S}$ and verifies these subshares by checking the degree of the polynomial and the constant term of the polynomial from which they came.
5. $\mathcal{F}$ uses the valid subshares to generate the new shares, $\{s_i'\}_{i=0}^{n_i}$, and sends s_i' to $p_i \in P_i$. It also sends s_i' to $\mathcal{S}$ for each corrupt p_i.

Security Proof: Recall that security is shown in the UC framework by specifying how $\mathcal{S}$ interacts with $\mathcal{Z}$ to make the real and ideal worlds indistinguishable. The set $\mathcal{L}_\pi$ from the real world is the same as in the HbC case. Recall that this set has four parts to it. The set

$\mathcal{L}_{\mathcal{F}}$ is also the same as it was before and has only two parts to it. The simulator generates the ideal world view of $\mathcal{L}_\pi$ using $\mathcal{L}_{\mathcal{F}}$ in the same manner as above with the exception of allowing $\mathcal{A}$ to choose the subshares of part 3. Some of these (potentially corrupted) subshares are used in part 4. To generate the remainder of part 4, $\mathcal{S}$ follows a process similar to what was done in the HbC model, but ignores any subshares that $\mathcal{A}$ corrupted.

Indistinguishability between the two worlds changes only slightly in the malicious case. Recall that $\mathcal{Z}$'s view in both worlds is defined by $\mathcal{L}_\pi$, which consists of the four parts presented earlier. The first part is the same as in the HbC model, so indistinguishability follows from that previous discussion. Thanks to the robust reconstruction done in the malicious model, the second part is also the same as in the HbC case, so again, indistinguishability follows from that discussion. The subshares generated by corrupt parties in P_j, which form the third part of $\mathcal{L}_\pi$, are, in both worlds, generated by $\mathcal{A}$, who is the same in both worlds, so they are indistinguishable. The fourth part consists of subsharings of the second part, with some of the subshares potentially corrupted by $\mathcal{A}$. Since $\mathcal{A}$ is the same in both worlds, using analysis similar to that done in the HbC case for part 4, this part is also indistinguishable. Therefore, since the sets $\mathcal{L}_\pi$ are indistinguishable between the two worlds, and the malicious transfer protocol is UC secure.

IV. Applications

This chapter describes two application areas to which T-MPC is applied. The purpose of this is to highlight the benefits that are gained by using T-MPC over MPC. Chapter 1 briefly introduced the application areas. The scenario and setup for each application is fully defined in this chapter. Prior and related work specific to each application is also discussed. This chapter then presents the application of T-MPC to the applications and presents how T-MPC benefits each.

4.1 Smart Grid

4.1.1 Motivation.

In 2010, the United States' Federal Bureau of Investigation (FBI) issued a report on smart meter hacking which detailed how one utility could to lose up to $400 million annually due to meter tampering attacks [52]. The report concludes that "this type of fraud will also spread because of the ease of intrusion and the economic benefit to both the hacker and the electric customer." The specific attack detailed in the report was an energy stealing attack, or one in which the meter misrepresents the consumption, resulting in a lower bill for the user. Other published attacks on smart meters aim to allow someone to read consumption information off of the meters. In one such attack, using just $20 in hardware, the researcher was able to read consumption information off all smart meters within range (about 19 meters) [53]. This second attack is a direct attack on privacy as one could use it to remotely learn another's consumption information. Published research has shown that consumption information can reveal very private information such as when a home is vacant, what appliances are being used, or whether or not expensive electronics might be present in the home [1, 2].

A number of privacy preserving data aggregation protocols have been proposed and studied for application to this problem (see [54] for an overview). Typically these protocols have focused on spatial (i.e., aggregating information from many meters at one instance in time) or temporal (i.e., aggregating information from one meter over a period of time) aggregation and have often focused on computing only the summation function. Secure multiparty computation (MPC) offers a capability to compute any function (including summation) while mitigating privacy risks. Due to its perceived complexity, MPC has not seen a lot of application into smart meter networks. Indeed many of the public-key based systems use additively-homomorphic ciphers (e.g., [55–59]). Interestingly generic MPC protocols are actually more efficient than existing protocols that use homomorphic encryption. T-MPC enables highly scalable and efficient computations.

To better motivate the utility of T-MPC, consider the following scenario. Let four parties, called Alice, Bob, Chuck and Doug, are interested in compute the sum of their inputs, x_a, x_b, x_c, x_d. Using traditional MPC, if Doug is unavailable, Alice, Bob and Chuck must wait until he is available to begin the computation. Clearly this is inefficient, as $sum(x_a, x_b, x_c, x_d) = sum(sum(x_a, x_b, x_c), x_d)$ and, therefore, Alice, Bob and Chuck could proceed with computing the sum of their inputs and, when Doug finally arrives, let him join in on the computation to add his input to get the final result. There are a number of technical limitations to applying MPC in this manner as traditional MPC assumes a static set of computation parties. T-MPC solves this problem by allowing partial computations from one set of parties (e.g., Alice, Bob and Chuck) to be transferred to a new set of parties (e.g., Alice, Bob, Chuck and Doug) without revealing intermediate computation values. The new set could be a superset, subset or completely independent of the original set of parties. This specific example of T-MPC generalizes to cases where f is composed of other functions, for example $f = g(x_1, x_2, x_4, h(x_2, x_3))$.

Another example of the utility of T-MPC is in optimizing certain computations. For example, consider a very large set of smart meters, say $n = 1,000,000$ and again $f = sum$. Using traditional MPC directly for this computation is very inefficient as all n must participate in all portions of the computation. Since many standard MPC protocols rely on secret-sharing inputs, such a computation would require each party to share its input with the $n - 1$ other parties. This is very inefficient and becomes unwieldy as n grows larger and larger. One technique commonly used to fix this inefficiency is to have the parties share their inputs with a small number of computation servers instead of all other parties. The computation servers carry out the computation of f on behalf of the parties. This technique, however, presupposes the availability of such servers and an infrastructure for communicating with these servers. In the smart grid, wireless sensor networks, and potentially many other applications, this assumption will not necessarily hold. T-MPC solves the problem for certain computations without the additional assumptions by allowing small groups to compute local results and securely transfer the computation. This is best illustrated by considering a tree structure as depicted in Figure 4.1.

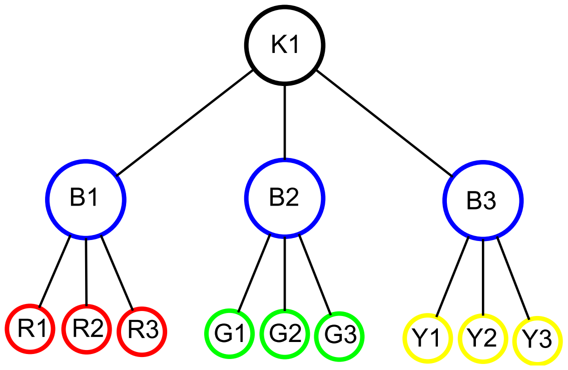


Figure 4.1: Applying transferable multiparty computation to a tree structured network.

Using T-MPC, parties R1, R2 and R3 can compute the sum of their inputs and transfer the result to B1, B2 and B3 without revealing their individual inputs or the intermediate sum. Similarly G1, G2, G3 and Y1, Y2, Y3 compute the sum of their inputs and transfer the result up the tree. When B1, B2 and B3 receive input from all of the parties below, they sum those values privately along with their own inputs and send the necessary information to K1 to reconstruct the result. More generically, siblings compute their portion of the computation, combine it with results from their children, and transfer the computation to the next level up in the tree. Being able to structure computations in this fashion leads to much more efficient computations while still providing privacy of individual inputs and intermediate results. T-MPC is orders of magnitude more efficient than simply using existing MPC protocols for in-network smart grid computations. For instance, for the standard deviation function and using a common MPC protocol, an example smart meter network with around 700 meters would take approximately fifteen minutes to complete the computation. On the other hand, using T-MPC, the standard deviation of over a million meter nodes takes just a few seconds to compute.

4.1.2 Experimental Setup.

In studying the application of T-MPC to the smart grid, assume a network organization similar to that of Figure 4.1. The analysis presented here loos at various branching factors and various numbers of nodes. Assume that nodes can communicate directly with their sibling nodes, their parent node, and their parent's siblings. Also assume that the root node of the tree can communicate with any other node in the tree (either with a fully-connected network or using routing). To simplify the analysis, assume that each level in the tree is complete.

In this section, T-MPC is compared with the honest-but-curious MPC protocol from [47]. In fact, since T-MPC uses a generic MPC protocol with the transfer protocol, the HbC T-MPC uses the MPC protocol from [47]. Note that the protocol from [47] is UC-secure and thus composes securely with the transfer protocol. The implementation used in the experiments comes from the VIFF (http://viff.dk) framework that runs on Python. The meter nodes are Gumstix Overo Earths with a 600MHz processor and 256MB of RAM. While this is more powerful than current smart meters, their performance may be found in future smart meters, possibly by using custom chips. The final result of the computation will be reconstructed by the root of the tree. This node is a laptop running an Intel Core i5-540M CPU and 4GB of RAM in the experimentations. For communications, assume a half-duplex, 250kbps wireless link with a single communications channel.

4.1.3 Timing.

To assist in the analysis the operations necessary for HbC MPC and malicious MPC are timed on both the Gumstix and the laptop. As **add** is a constant time operation, it is measured by the average time to add two shares in VIFF. The operations **share**, **recombine**, and **mult** take linear time in the number of parties, thus, they are estimated by calculating the coefficients of the line by running computations with various numbers of parties and run linear regression analysis. Table 4.1 shows the computed values for each operation for the honest-but-curious MPC protocol. For the linear operations, the coefficients of the line $y = c_2x + c_1$ estimate the time to compute each operation for a given number of parties. Malicious model timing was computed similarly, but the preprocessing operation **genTriple**, which is quadratic and is estimated using a quadratic function $y = c_3x^2+c_2x+c_1$. This information is shown in Table 4.2. These timing values assist in computing the timing for the additional T-MPC operations **transfer** and **recombine_transfer** as both operations are composed of the operations already specified. The time to run **transfer** in both models is simply the time to run **share** in the respective model. For HbC T-MPC, the time to run **recombine_transfer** is simply the time to run **recombine**, while for malicious T-MPC the Reed-Solomon decoder is used, which, for small numbers of parties, the fastest method is brute force, which has factorial complexity.

Table 4.1: Timing estimation for HbC MPC.

	share		**recombine**		**add**	**mult**	
	c_2	c_1	c_2	c_1	c_1	c_2	c_1
Meter	1.178	−1.019	0.056	0.289	0.07	1.234	−0.66
Sink	0.043	−0.036	0.003	0.01	0.002	0.049	−0.026

Table 4.2: Timing estimation for malicious MPC.

	genTriple			**share**		**recombine**		**add**	**mult**	
	c_3	c_2	c_1	c_2	c_1	c_2	c_1	c_1	c_2	c_1
Meter	0.141	4.879	−3.209	1.178	−1.019	0.056	0.289	0.07	0.112	0.58
Sink	0.002	0.181	−0.114	0.043	−0.036	0.003	0.01	0.002	0.006	0.02

4.1.4 Analysis.

The analysis in this section uses the timing values from Tables 4.1 and 4.2 to compute the total time to execute two functions of interest, namely, summation and standard deviation. To avoid division and square roots in the computation of the standard deviation $\left(\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2}\right)$, the parties compute only the numerator (it is assumed that the root of the tree knows n). Note that this leaks no additional information when compared to computing the standard deviation in its entirety.

T-MPC allows for parallelizing in-grid computations using the tree structure previously described. This results in a significant optimization of both the sum and standard deviation. Figure 4.2 shows the time to execute each computation for a fixed network tree branching factor (i.e., the number of children that each node has) of 10. The benefit of T-MPC is clear. Using the MPC protocol, one can compute the sum and standard deviations in less than fifteen minutes as long as there are fewer than 2647 and 777 meter nodes, respectively. Compare this with T-MPC, however, where even at 10^6 meter nodes, it takes well under under the fifteen minute mark to run the computation. In fact, the protocol execution time with this large of a network takes less than 2 seconds for both the sum and the standard deviation. Thus, T-MPC greatly enhances the scalability of the network. Furthermore, this increased scalability allows for more fine-grained information reporting in reasonably sized networks. This is very significant as prior, specialized summation protocols which use homomorphic encryption can only support networks up to 50 nodes at an information reporting granularity of 60 seconds [39].

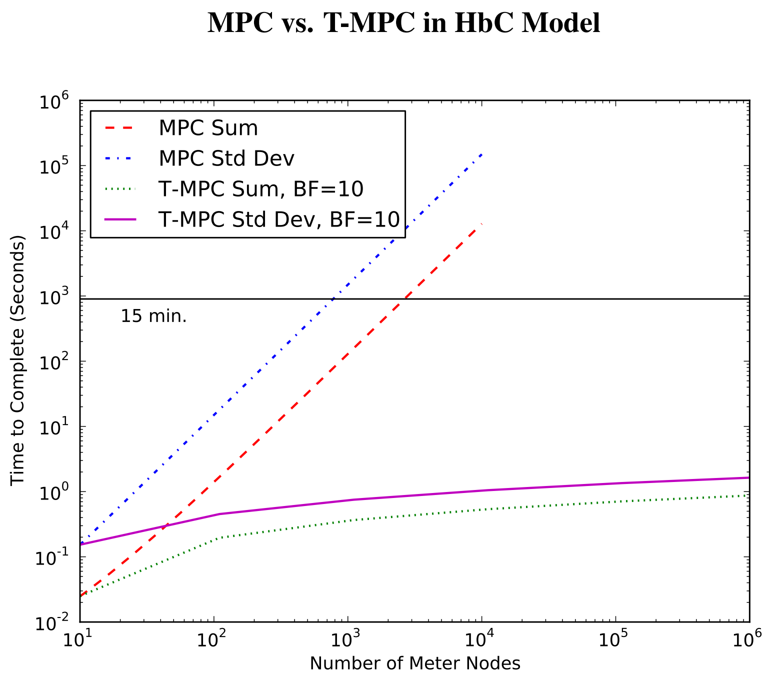


Figure 4.2: MPC vs. T-MPC (tree branching factor of 10), Honest-but-Curious Model.

The comparison between MPC and T-MPC used a branching factor of 10 in the construction of the tree. The branching factor determines a number of items including the number of shares used when secret sharing inputs, the depth of the tree, amount of parallelism, to name a few. Figure 4.3 plots the time to execute T-MPC for both sum and standard deviation with various branching factors. From the figure, note that branching factor does play some role into the efficiency of the protocol. The differences are fairly minor however. As the branching factor approaches *n* (the total number of meter nodes), however, T-MPC in this case becomes equivalent to the underlying MPC protocol. There is another underlying aspect to the branching factor. For a subset of parties P_i of size n_i there are $t_i < n_i/2$ adversary-controlled parties due to the underlying MPC protocol. In the case of a tree structured network $n_i = \text{BF}$. Therefore, for BF $= 3$ there can only be one corrupted party in each subset but for BF $= 11$ the T-MPC protocol could handle up to five corrupted parties. Understanding the implications of this is especially important in real-world deployments.

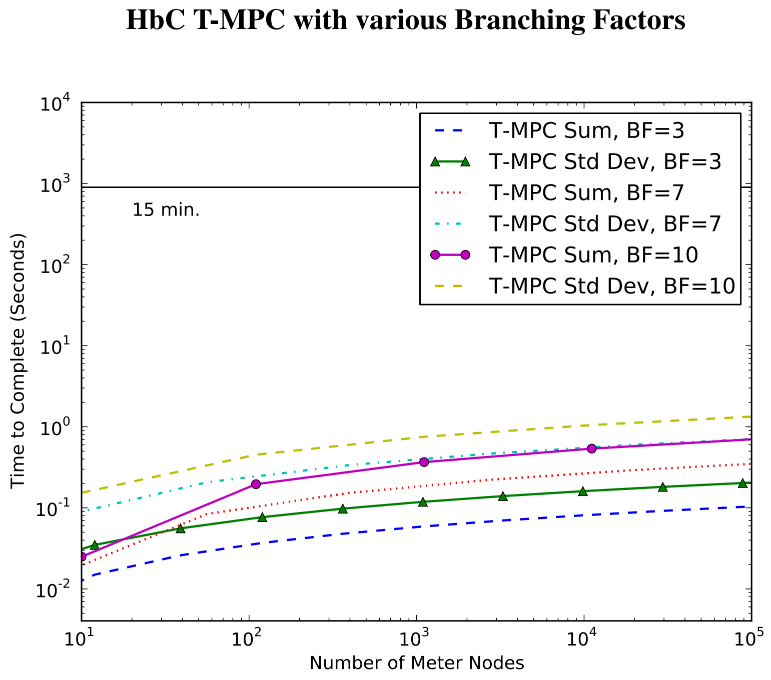


Figure 4.3: Various branching factors (BF) for Transferable Multiparty Computation, Honest-but-Curious Model.

The results of the malicious model experiments are very similar in that T-MPC in the malicious model is far more efficient than MPC. Figure 4.4 shows the comparison between T-MPC and MPC while Figure 4.5 shows how the branching factor affects execution time.

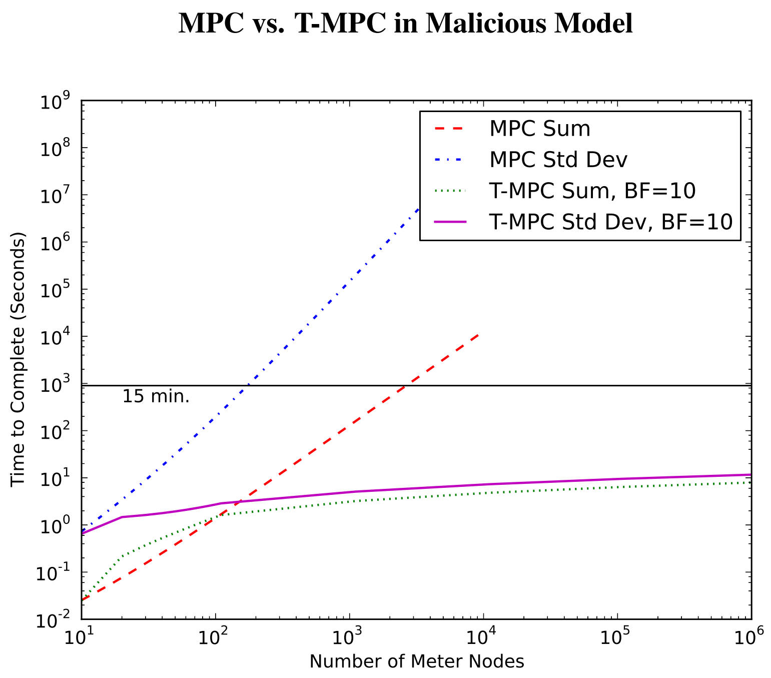


Figure 4.4: MPC vs T-MPC (tree branching factor of 10), Malicious Model.

4.2 Decentralized Reputation Systems

4.2.1 Motivation.

Reputation in another party is a measure of confidence that that party will conform to a certain behavior or perform a certain action. For example, consider a mobile ad-hoc network (MANET) in which a party's neighbors are used to route messages. A party might build up reputation information on his neighbors by observing whether or not they forward messages he sends to them. As new parties join the network, however, they will have no reputation information on others in the network. A reputation system can be used to help bootstrap this information. In a typical reputation system, a party can ask others for their reputation scores on a particular party, and then use, for example, the average of the responses to bootstrap their own reputation information.

Many online marketplaces have reputation systems built in. They allow users to provide feedback (or ratings) on products and vendors. The aggregate of this feedback

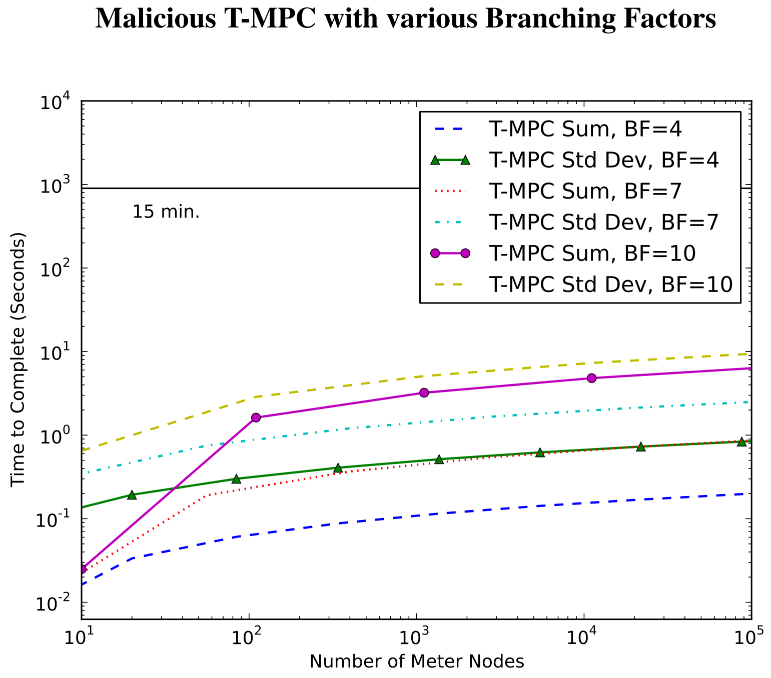


Figure 4.5: Various branching factors (BF) for Transferable Multiparty Computation, Malicious Model.

information is displayed to customers in order to help them make choices about what product to purchase or from whom to purchase the product. These are examples of centralized reputation systems. These reputation systems can function because the market operator (e.g., Amazon or eBay) is at least somewhat trusted by both vendors and consumers. Indeed, it is in the market operator's best interest to provide honest feedback to customers.

In many scenarios, however, such a trusted party does not exist. This includes peer-to-peer systems, MANETs, and others. For this reason, decentralized reputation systems (DRS) exist. Example systems can be found in [60–63], which are applied to both peer-to-peer systems and MANETs. These systems are more ad-hoc in nature. In these systems, a party p_q, called the querying party, would like to interact with another party p_t, called the target party, but p_q has no reputation information on p_t. Therefore, p_q forms a set of parties, U and asks each party in U to provide their reputation information on p_t. p_q then averages these and stores the result. The result is used to help p_q know whether or not to interact with p_t.

Recently, researchers have become concerned about privacy issues in DRS. In particular, if privacy of reputation information is not maintained, parties providing reputation information to a query could be subject to retaliation, retribution, or attack. Therefore, it may be in a party's best interest to not provide honest feedback. To alleviate this situation, researchers have proposed a number of privacy-preserving decentralized reputation systems (PDRS). In such systems, instead of providing their reputation information directly to p_i, the parties in U run a protocol which allows them to jointly compute a function of each of their individual reputation values about p_k (typically they compute the sum) and then reveal the result of the computation to p_i. The protocol run by the parties is specifically designed so that they have strong assurances that their reputation information has been kept private. Examples of such can be found in [64–68].

All existing PDRS fall into the category of static PDRS. Static means that when a party leaves the network, all of the reputation information they have built up through interactions with others in the network leaves with them. In situations where reputation information is sparse, however, this can be a big problem. The security of these systems is often based on there being a sufficiently large number of parties in the query set U, some fraction of which must be honest. This section presents a dynamic, privacy-preserving decentralized reputation system (Dyn-PDRS) as a solution to this problem. A Dyn-PDRS enables parties to run a delegation protocol when they want to leave the network. In this protocol, they delegate their reputation information to a set, D, of other parties in the network. The delegation is done in such a way that the party's privacy is still maintained. That party is then free to leave the network. When that party appears in a query set U, the parties in D are able to act on its behalf. Furthermore, a Dyn-PDRS provides a redelegation protocol which is run when a party in D wants to leave the network. This allows the parties in D to redelegate to a new set D'. That set D' can then act on the original party's behalf.

This section presents the following contributions.

1. A description of existing PDRS from the literature and an illustration of how these systems fail when parties are allowed to leave the network.

2. A more formal definition of PDRS and Dyn-PDRS and a description of the problem setting.

3. Four protocols for building a Dyn-PDRS. The first is necessary for a PDRS and is similar to existing work in the area. The next three are necessary to build the Dyn-PDRS.

4. Correctness and security analysis of the protocols.

The protocols are secure in the honest-but-curious or semi-honest adversary model [40]. This model assumes that corrupt parties execute the protocol as specified, but use any information gleaned during execution to attempt to violate another party's privacy. This section presents results of a number of simulations to illustrate the benefits of a Dyn-PDRS over the traditional PDRS. While the delegation and redelegation protocols given can be run indefinitely, it turns out this can have a major impact on security. Section 4.2.6 formalizes the issue and presents two delegation strategies for dealing with it. Section 4.2.7 describes an implementation of the protocols, and Section 4.2.8 describes timing experiments conducted using the implementation.

4.2.2 Related Work.

A number of protocols have been proposed to construct PDRS. This section describes some of the prominent ones and comments on why the problem of operating in networks where parties are constantly leaving and rejoining the network is a concern. The description focuses on protocols which are secure in the honest-but-curious model as that is the model used in this section.

4.2.2.1 Pavlov et al.

One of the earliest works in privacy-preserving decentralized reputation systems comes from Pavlov et al. [64]. An important proof coming from this work is that if the querying node is corrupt, there must be at least two honest nodes or privacy cannot be achieved. The authors also present three protocols (of varying strengths and security guarantees) which enable such a system. Their second protocol is closest to the present setting (full-threshold security where corrupt parties are allowed to collude) and is described below. The querying party begins the protocol by running a witness selection scheme. This results in a set of witnesses who will provide feedback on the target party and, with high probability, will have at least two honest witnesses. The querying party sends a description of the set to all parties in the set. Each witness splits his reputation score on the target party using additive secret sharing and sends one share to each party in the protocol (including the querying party) and keeps one share for himself. Once a party has gathered shares from every other party, he sums them all up and sends the result to the querying party. The querying party then sums all the values he receives to recover the sum of the reputation values. For security and correctness proofs of this protocol, see the original work by Pavlov et al.

In the case of dynamic networks, the problem with Pavlov's protocol is that, while honest parties which could provide feedback for a particular target party will come and go due to normal churn in the network, dishonest parties will not necessarily follow this pattern, making them more likely to be chosen as witnesses. Pavlov et al. prove their witness selection scheme will result in a witness set with at least two honest witnesses with probability greater than $(1 - \frac{1}{n})(\frac{N-b-1}{N-1})$, where n is the number of witnesses, N is the number of possible witnesses (i.e., the number of parties with reputation information on the target), and b is the number of corrupt parties.

For Pavlov's protocol, a dynamic network has the effect of lowering N while b remains constant. Figure 4.6 shows how this affects the probability of having at least two honest witnesses. The probability for a hypothetical static network is also shown in the figure for reference. Here, the fraction of corrupt parties is fixed at 0.1 and the size of the witness set is one-tenth of the original network size. It is clear that the dynamic nature of the network has a significant impact on the security of Pavlov's protocol.

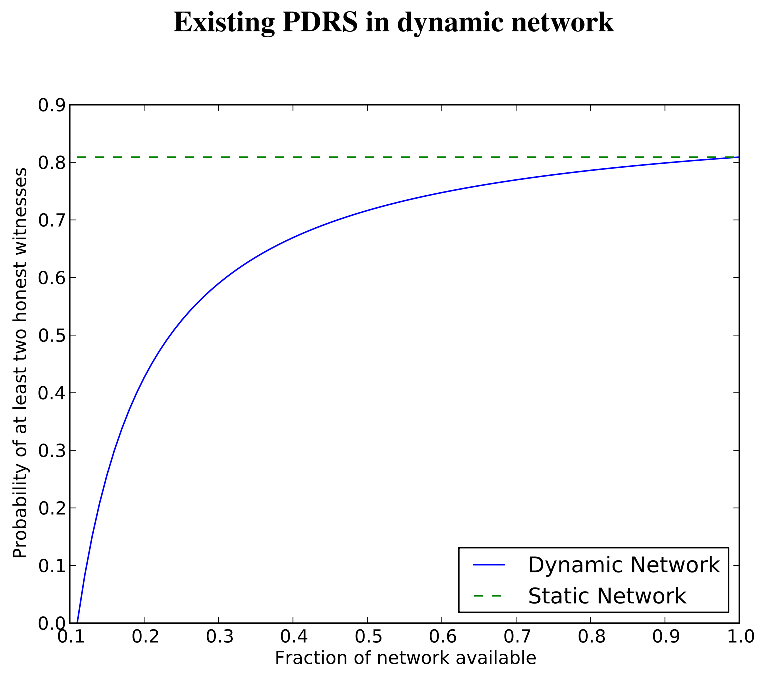


Figure 4.6: Comparison of security for Pavlov's protocol [64] in static vs. dynamic networks.

4.2.2.2 Hasan et al.

Hasan et al. [65] propose the k-shares reputation protocol which builds upon the work of Pavlov et al. The benefit of the k-shares protocol is that witnesses are able to maximize and quantify the probability that their reputation information is kept private. In this protocol, the querying agent chooses a set of witnesses (the exact method for this is not specified in the paper). The description of the set of witnesses is sent to each witness. Each witness chooses a subset of the witnesses of size up to k which he considers trustworthy. The witness then shares their reputation information with the subset using additive secret sharing and sends the description of the subset to the querying party. The querying party informs each witness who they will receive shares from. Each witness, upon receiving shares from other witnesses, sums them up and sends the result to the querying party. The querying party sums all of these values to get the sum of the reputation values. Note that if a witness decides to, he may choose not to input his reputation information if he does not trust enough parties in the witness set. Furthermore, since each witness is selecting up to k other witnesses that he trusts, the authors note that this leaks some information about trust relationships (but not specific reputation information). The authors propose solving this by allowing the querying party to add a few untrusted parties to the subset and then selecting the same subset for repeated queries.

Consider the operation of Hasan's protocol in a dynamic network. As the authors do not specify how the set of witnesses is chosen, assume it happens in the same manner as in Pavlov's protocol. In Hasan's protocol, as fewer and fewer honest witnesses are available, the remaining honest witnesses will likely refuse to take part in the computation, thus preserving their privacy. Note, however, that the fact that more honest parties are refusing to participate in reputation computations is not a good thing for the system as a whole. Another issue arises when attempting to use Hasan's protocol in a dynamic network. In order to provide high efficiency, the authors require that $k << n$, where n is the size of the witness set and k is the maximum size of the subsets chosen by each witness. In a dynamic network, it is possible that this inequality cannot be met as the number of available (i.e., currently part of the network) witnesses could be much smaller than in an entirely static network.

4.2.2.3 Other Protocols.

A number of other decentralized, privacy-preserving reputation systems have been proposed in the literature (e.g., [66, 67]). Of the other protocols available, all have similar issues with regards to dynamic networks. In particular, the fact that reputation information from trustworthy parties may not be available at query time impacts the security of existing reputation systems. This illustrates the importance of availability in decentralized reputation systems in general. The solution to the problem is non-trivial as privacy of reputation information used to help the querying party compute a reputation value for the target party must also be preserved.

4.2.3 Problem Setting and Definitions.

The problem area is that of computing reputation in a privacy-preserving manner, in dynamic, decentralized networks. This section defines the working environment and other important details of the setup. Some details are abstracted in order to focus on building solid protocols to enable such a reputation system.

Let P be the set of parties which form the network. Parties in P may leave and join the network as they please. Assume that each pair, $p_i, p_j \in P$, is connected by a secure, authenticated channel. Party p_i stores reputation information that it has generated about another party p_j, say v_{ij}. Let v_{ij} be between 0 and some global maximum reputation v_{max} if p_i has reputation information on p_j, otherwise, $v_{ij} = \perp$.

Decentralized reputation systems are useful in the case where p_i needs to interact with some p_k but $v_{ik} = \perp$. In this case, p_i forms a set $U \subset P$ and queries parties in U about p_k to help it compute v_{ik}. For example, if $v_{ik} = \frac{1}{|U|} \sum_{p_j \in U} v_{jk}$ the system is additive. Such a system is also privacy-preserving if it fits the following definition.

Definition 1 (Privacy-preserving Decentralized Reputation System (PDRS))**.** *A (additive) PDRS consists of a decentralized protocol π_{add} which allows a querying party, p_i, to* *compute* $v_{ik} = \frac{1}{|U|} \sum_{p_j \in U} v_{jk}$*, without any of the v values being leaked to any other party. Here U is the query set and is chosen by* p_i*.*

Definition 2 (Additive Secret Sharing)**.** *Let* $\mathbb{G}$ *be a cyclic group. The additive secret sharing of* $s \in \mathbb{G}$ *are the shares* $s_1, \ldots, s_n \in \mathbb{G}$ *such that*

1. $s = s_1 + s_2 + \cdots + s_n$ *and*
2. $s_1, s_2, \ldots, s_{n-1}$ *are chosen at random from* $\mathbb{G}$ *and*
3. $s_n = s - (s_1 + s_2 + \cdots + s_{n-1})$*.*

Let $S_n : \mathbb{G} \to \mathbb{G}^n$ *be the additive secret sharing function which outputs n shares of the input, that is,* $S_n(s) = (s_1, \ldots, s_n)$*. Let* $S_n(s)[i]$ *be the i-th share of s. Given the n shares, one can reconstruct s simply by adding the shares together.*

Additive secret sharing, defined above, has been used in a number of general secure multiparty computation protocols as a way to preserve privacy [43, 69], and is used the Dyn-PDRS presented in this section. It is linear, i.e, given shares of two values, one can compute a share of the sum of those values without inverting the sharing function, or mathematically

$$S_n(s)[i] + S_n(s')[i] = S_n(s + s')[i].$$

Furthermore, any adversary who does not know all the shares cannot compute the secret. In fact, an adversary with up to $n - 1$ shares gains no additional information about s. In other words, additive secret sharing is information-theoretically secure. The subscript is omitted when it is clear from the context.

Definition 3 (Dynamic, Privacy-preserving Decentralized Reputation System (Dyn-PDRS))**.** *A (additive) Dyn-PDRS consists of a protocol* π_{add} *as in Definition 1 and three additional protocols:* π_{del}*,* π_{act} *and* $\pi_{re\,del}$*. Where* π_{del} *allows a party to delegate the reputation information it holds to a set of parties D while still preserving the privacy of that information. The* *protocol π_{act} allows a set of parties who have been authorized to act on another party's behalf to enter that party's information into the protocol π_{add} while still preserving the party's privacy. Finally, the protocol $\pi_{re\,del}$ lets a set of parties, say D, re-delegate reputation information that was delegated to it to another set of parties, say D′, in a way which maintains the privacy of the information.*

The next section gives specific instances of these protocols and how they are composed to form a Dyn-PDRS. There is some tradeoff to be balanced in delegation. Section 4.2.6 explores delegation strategies in order to balance the tradeoff between information availability and privacy.

4.2.4 Protocols.

This section presents the four protocols introduced earlier. This section first presents π_{add}, the protocol to allow p_i to use the set U to compute $v_{ik} = \frac{1}{|U|}\sum_{p_j \in U} v_{jk}$ privately. The summation is computed via a simple multiparty computation built on additive secret sharing. The concept is similar to previous work in decentralized reputation systems and has similar performance characteristics. π_{add} by itself could be used as the basis of a PDRS. It is important to note that in a Dyn-PDRS, since delegations are allowed, the set U may contain parties which are not currently online, as long as the party has delegated its reputation information. The set U can be generated using methods from prior work, for example, Pavlov's witness selection protocol. All parties in U must have reputation information on the target, p_k.

Next, this section presents the remaining three protocols, π_{del}, π_{act} and $\pi_{re\,del}$ to enable privacy-preserving delegation. Together, these protocols enable a reputation system where parties can leave the network, yet delegate their reputation information in such a way that it can still be used to assist other parties in computing reputation. Some details of the underlying communication system and about how the protocols interact are kept abstract to keep the discussion focused on the protocols themselves. Section 4.2.7 describes the implementation of these protocols in a real system and describe these parts in more detail. In a general sense, the security of the protocols are secure for up to $n - 1$ corrupt parties. However, due to the specific computation, summation, $n - 1$ corrupt parties can learn the remaining honest party's input by subtracting their individual inputs from the output. Only the querying party should learn the output, so if the querying party is corrupt, there must be at least two honest parties in U. For simplicity the protocols are presented as if the querying party is honest. In the case of a dishonest querying party, the only thing that changes is the number of corruptions tolerated.

4.2.4.1 The PDRS Protocol.

Let p_i be the querying party, who wants to compute v_{ik} for some party p_k. Let U be the set of witnesses with inputs to the computation. The protocol π_{add} is shown in Protocol 1. Note that while not identical to previously proposed protocols for PDRS, the protocol is very similar, and, taken in its own right, should have similar performance.

Correctness: The correctness of the protocol is guaranteed due to the linear nature of additive secret sharing. Mathematically, $\sum_{p_j \in U} S(v_{jk})$, where addition is performed point-wise on the sharing vectors, is equal to $S(\sum_{p_j \in U} v_{jk})$. These shares are, in essence, what the parties in U send to p_i in the next to last step. So,

$$v_{ik} = \frac{1}{|U|} \sum_{p_j \in U} t_j = \frac{1}{|U|} \sum_{p_j \in U} v_{jk}.$$

Security: The security of the protocol comes from the security guarantees of additive secret sharing. As long as the adversary has not corrupted all of U, all of the individual reputation values v_{jk} as well as the output value v_{ik} are kept private.

4.2.4.2 The Dyn-PDRS Protocols.

Say party $p_\ell \in P$ is leaving the network. In order to not lose all the reputation information of p_ℓ, this section, proposes the necessary protocols to allow p_ℓ to delegate its reputation information to a set of parties $D \subset P$. This includes a protocol to allow the parties in D to act on behalf of p_ℓ whenever p_ℓ appears in a query set U, a protocol to

Protocol 1 π_{add}.

1. p_q sends the description of the set of witnesses U and the identity p_t to each party in U.

2. Each $p_j \in U$ computes $(s_1, \ldots, s_{|U|}) = S_{|U|}(v_{jt})$ and sends one share to each other party in U and keeps one share for himself.

3. Each p_j collects one share from each of the other parties in U. Let $(r_1, \ldots, r_{|U|})$ be the shares p_j collects (including his own share).

4. Each p_j then computes $t_j = r_1 + r_2 + \cdots + r_{|U|}$ and sends t_j to p_i.

5. Party p_i receives $|U|$ shares, t_j from $p_j \in U$, and sets $v_{it} = \frac{1}{|U|} \sum_{p_j \in U} t_j$.

allow the parties in D to transfer the delegation of p_ℓ's reputation information to a new set D'. This protocol is used when a party in D is leaving the network. D and D' may be of different sizes, overlap or be completely independent. When p_ℓ rejoins the network, the parties in D can simply discard p_ℓ's reputation information. It would be fairly simple, however, to also allow the parties in D to return the reputation information back to p_ℓ. For now, let the set D be chosen at random. Section 4.2.6 explores other methods for choosing D and the redelegation sets.

Protocol 2 describes π_{del}. The correctness and security of this protocol come directly from the correctness and security of additive secret sharing, discussed earlier. As long as the adversary does not control all of the parties in D, p_ℓ's reputation information is kept private.

Once the parties in D have received the information sent by p_ℓ in Protocol 2 and verified the digital signature, they are ready to act on his behalf. At some later point in time

Protocol 2 π_{del}.

1. p_ℓ chooses a set of delegates $D \subset P$.
2. For each $p_j \in P$ where $v_{\ell j} \neq \bot$

   p_ℓ computes $shares_j = S_{|D|}(v_{\ell j})$ and sends the identity j and one share to each party in D.
3. p_ℓ digitally signs a message signifying that it has delegated its reputation information to the set D and sends the message and signature to each party in D .

they will see a query set U that contains p_ℓ when a party, say p_i, initiates Protocol 1. At this point the parties in D run π_{act}, shown in Protocol 3.

At the end of Protocol 3, the parties in U' can complete the execution of Protocol 1. Some interesting features of the protocol are that not all of D is required to participate in the execution of Protocol 1 and that the trust value $v_{\ell k}$ is never revealed, either to the parties in D or the parties in U'.

Correctness: From Protocol 2, the parties in D hold shares of the reputation value $v_{\ell k}$, say $shares_k = (d_1, \ldots, d_{|D|})$ where $v_{\ell k} = d_1 + \cdots + d_{|D|}$. These shares are then split into subshares and distributed to the parties in U'. In other words, d_j is split into $d'_{j1}, \ldots, d'_{j|U'|}$. Notice that the sum of all the subshares for every d_j is still $v_{\ell k}$. One subshare of each d_j is sent to one party in U'. Since addition is commutative, it turns out that the sum of all the r_ℓ shares computed in Step 6 of the protocol is still $v_{\ell k}$. Thus, the parties in U' have additive shares of $v_{\ell k}$ as needed for the protocol to be correct.

Security: Protocol 3 is secure from the perspective that it does not give the adversary any additional information about $v_{\ell k}$. This is shown in the worst case, i.e., when the adversary controls all parties but one in D and all parties but one in U'. Security in the

Protocol 3 π_{act}.

1. The parties in D notify the parties in U that they are to act on behalf of p_ℓ by sending them the message and digital signature received from p_ℓ.

2. Parties in D select one of them to take p_ℓ's place in the set U and notifies the parties in U of this choice.

3. The parties in U validate the digital signature and replace p_ℓ in the set U with the party chosen in the previous step. Call this new set U'.

4. The parties in U use the set U' for sharing when continuing Protocol 1 with the exception of computing r_ℓ (the input shares that would have come from p_ℓ).

5. Each party in D takes its share of $shares_k$, say s_k, received in Protocol 2 and computes $(s_1, \ldots, s_{|U'|}) = S_{|U'|}(s_k)$ and sends one share to each party in U'.

6. Each party in U' receives $|D|$ shares from the previous step. Call these shares $(s'_1, \ldots, s'_{|D|})$. They then compute $r_\ell = s'_1 + \cdots + s'_{|D|}$. r_ℓ takes the place of what they would have received from p_ℓ in Step 3 of Protocol 1.

case that the adversary controls fewer parties is an immediate consequence from worst case security. Let $p_h \in D$ and $p'_h \in U'$ be the honest, uncorrupted parties in each set. Note that p_h and p'_h could be the same party. In the protocol, p_h will create a number of subshares, one of which will be sent to p'_h. Since the adversary will not know that share, due to the security of additive secret sharing, the adversary will also not know the r_ℓ that p'_h computes in Step 6 of the protocol. Without that value, the r_ℓ values computed by the corrupt parties give the adversary no additional information about $v_{\ell k}$. This shows that as long as there is at least one uncorrupted party in D and U', the protocol leaks no additional information about the private trust information.

If a party in D leaves, the remaining parties would not be able to act on p_ℓ's behalf. Therefore, before any party in D leaves the network, $\pi_{re\,del}$ is run, as shown in Protocol 4. Let $p_{\ell'} \in D$ be the party that is leaving the network. Furthermore, recall that from Protocol 2, the parties in D hold a number of pairs (j, s_j) where j is an identity of a party and s_j is a share of $v_{\ell j}$. How the set D' is chosen will be explored in a later section. For now, assume that D' is a new random set. The description of $\pi_{re\,del}$ focuses on the case where only one party has delegated information to the set D. The protocol can easily be adapted to the case where multiple parties have delegated to D by running it once for each party that has left the network and delegated to D.

Thus, by doing something similar to what was done in Protocol 3, i.e., creating and distributing subshares, the parties in D are able to transfer all delegated information they hold for p_ℓ to the set D' without revealing the values. Given the results of this protocol, simple modifications can be made to Protocol 3 to allow the set U to properly validate that D' is authorized to act on p_ℓ's behalf. Correctness and security proofs for this protocol follow a similar logic that was used in Protocol 3.

4.2.5 Simulation.

The protocols shown in the previous section can be used to build a dynamic, privacy-preserving decentralized reputation system (Dyn-PDRS). This section shows the utility of increasing availability in decentralized reputation systems through a number of simulations. In order to establish a comparison with previous work, simulation for no delegation of reputation information is also given. This is what all previous privacy-preserving decentralized reputation systems do. Thus, in a network with churn, the reputation information of parties leaves when the parties leave the network. Table 4.3 summarizes all the symbols used in this section.

Protocol 4 $\pi_{re\,del}$.

1. $p_{\ell'}$ sends a message to all other parties in D that it is leaving the network.

2. The parties in D select a new set D' which will be responsible for acting on behalf of p_{ℓ}.

3. Each party in D creates subshares of each s_j it holds and distributes one subshare to each party in D' along with the identity j.

4. For each j, each party in D' receives one subshare s_j from each party in D and stores the sum of these subshares along with j. The sum of these subshares is a new share of $v_{\ell j}$.

5. Parties in D also send the message and digital signature they received from p_{ℓ} to the parties in D'. They also each digitally sign a message stating that they are transferring delegation of p_{ℓ}'s reputation information to D'.

4.2.5.1 The Setup.

For the simulation, let N be the total number of parties in the network, a be the probability that a party is in the network during one iteration of the simulation ($1 - a$ is the probability that they are not in the network). Let c be the fraction of corrupt parties. The reputation system initializes by giving each party some reputation information on other parties in the network. Let b be the fraction of parties for which a given party holds reputation information at the start of the simulation.

At each iteration of the simulation some fraction, q, of the parties in the network ask for reputation information on some other in-network party. Also, in each iteration, some fraction, γ, of the network leaves or rejoins the network. Parties (both those in the network and those out of the network) will leave the network with probability $1 - a$

Table 4.3: Table of symbols for simulator.

Symbol	Description
N	Number of parties in the network
a	In-network probability
c	Fraction of corrupt parties
b	Fraction of information to bootstrap
q	Fraction that query in an iteration
$\|D\|$	Cardinality of the delegation set D
δ	Bound on delegation chain depth
γ	Network churn rate

or join the network (if they were already gone) with probability a. With $a = 1$ a static network is achieved. As seen previously, when a party leaves the network, they delegate their reputation information to a delegation set. If someone in that set leaves before the original party returns to the network, a redelegation occurs. Let δ be the bound on the total number of delegations and redelegations. Bounding depth of the delegation chain affects both efficiency and security. Let δ be the bound on the depth of the delegation chain. In other words, if $\delta = 1$, when p_ℓ leaves the network, he will delegate his trust information to some set D. When one of the parties in D leaves the network, they do not do any further delegations. With $\delta = 2$, p_ℓ would delegate to a set D who, in turn, would delegate to a set D' when a party in D is leaving the network, but the chain would end there. When p_ℓ returns to the network the delegation chain resets (i.e., delegation would occur again if p_ℓ left again). In effect, existing privacy-preserving decentralized reputation systems have $\delta = 0$, i.e., no delegation.

4.2.5.2 Varied δ.

This section shows how δ affects the level of information availability achieved by the Dyn-PDRS. Figure 4.7 shows a simulation with $N = 10000$, $a = 0.75$, $c = 0.2$, $b = 0.05$, $q = 0.05$, $|D| = 5$, $\gamma = 0.25$ and various values for δ. The static network represents the theoretical upper bound, for reference. Information availablity is the fraction of information available by counting the total number of reputation values available in the network (either directly from a party or through delegation) divided by the total possible number of reputation values ($N^2 - N$).

Dyn-PDRS Simulation for various δ

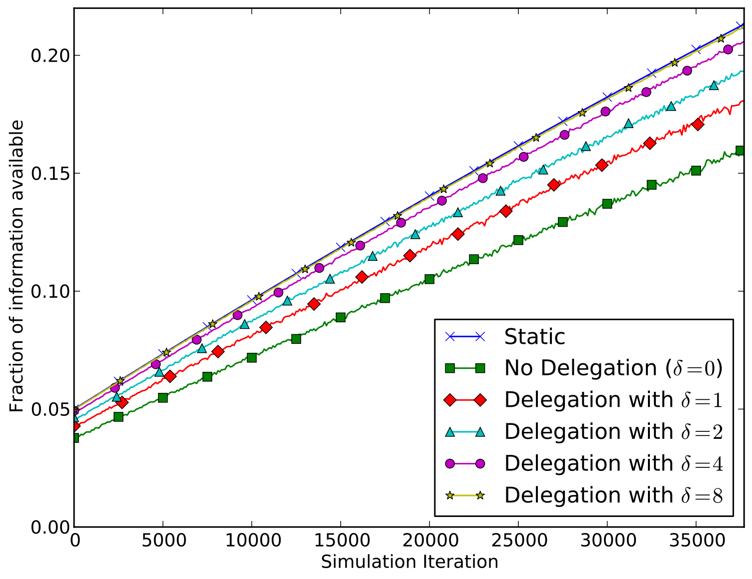


Figure 4.7: Information availability for various delegation depths (a = 0.75).

The figure shows that even with $\delta = 1$ there is a significant increase in the amount of available information. Furthermore, with $\delta = 4$ the information availability in the simulated Dyn-PDRS is very close to that of the fully static network. This plot illustrates how effective simple delegation can be in a reputation system.

Dyn-PDRS Simulation with decreased a

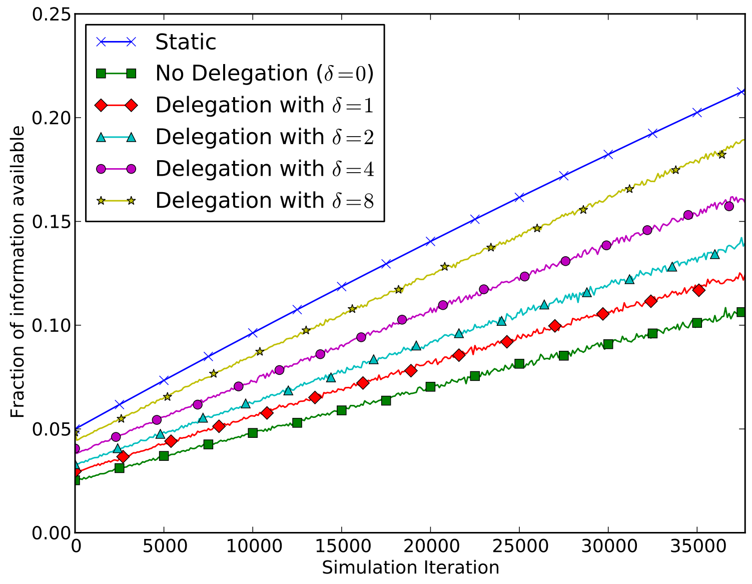


Figure 4.8: Information availability for various delegation depths (a = 0.50).

4.2.5.3 Varied a.

This section describes how changing a affects information availability. Figure 4.8 repeats the previous simulation but lowers a to 0.5. There is still a significant advantage in delegating reputation information, but it takes a longer delegation chain to approach the static system. In essence, the effect of a lower probability of availability of the parties is a slower growth of information availability in the system over time. To combat this in a deployed system, a deeper delegation chain can be used.

4.2.5.4 Varied γ.

The next simulation focuses on various values for γ. Recall that γ specifies what fraction of the parties might change their network status. This relates to the churn of the network. At each iteration of the simulation, γN of the parties will flip a weighted coin to determine if they should be in the network (either join the network if they were out, or stay in). a specifies the probability that the party should stay in or join the network. Other parameters are fixed at $N = 10000$, $a = 0.75$, $c = 0.2$, $b = 0.05$, $q = 0.05$, $|D| = 5$ and $\delta = 4$. The simulation uses the following values for γ: 0.25, 0.50, 0.75, 1.00. The results are shown in Figure 4.9. The plot includes the line for the static network for reference.

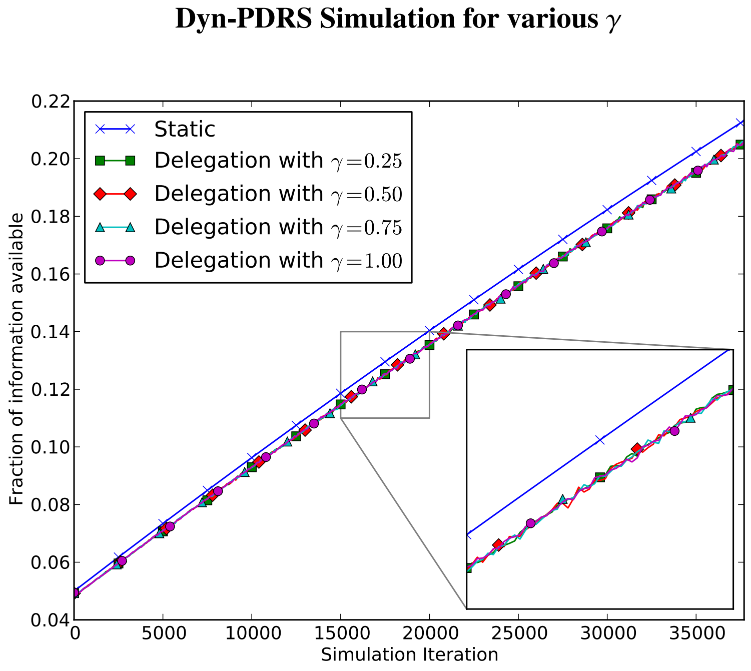


Figure 4.9: Information availability plot with various churn rates.

The figure that, surprisingly, the churn rate has little effect on information availability. To understand why this is the case, consider what happens when γ is high. Parties are more likely to leave the network, so they will have to delegate their reputation information. They are also, however, more likely to come back quickly, which means the delegation chain limit is less likely to be reached. Contrast this with the case where γ is low. Parties are less likely to leave the network, so they will not have to delegate their information as often. When they do leave, however, they stay out longer. But, since the parties in their delegation (and redelegation) set are less likely to leave also, the delegation chain will not grow as quickly.

Dyn-PDRS Simulation for various $|D|$

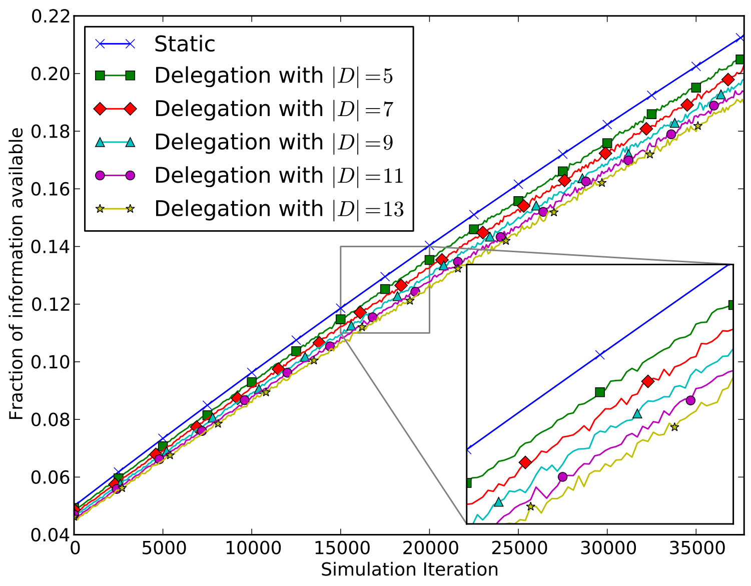


Figure 4.10: Information availability plot with various delegation set sizes.

4.2.5.5 Varied $|D|$*.*

This section describes the effect of the size of the delegation and redelegation sets on information availability. For simplicity, assume that the size of the delegation set and the redelegation sets are the same. Figure 4.10 shows the results of this simulation. This plot also shows the static network for reference. The figure shows that the size of the delegation set indeed has an effect on information availability. The effect is not drastic, but at the same time is non-negligible. The reason for lower information availability as $|D|$ increases is that there are more parties who can cause redelegations, thus, it is more likely that the delegation chain depth limit is reached.

4.2.5.6 Discussion.

From the previous simulations, it is clear that different parameters affect information availability differently. Network churn has little to no effect on information availability, but the in-network probability and the delegation chain depth can both have significant impacts.

Increasing δ increases the y-intercepts and the slopes of the lines in Figure 4.7. Comparing that figure with Figure 4.8, one can see that decreasing a decreases the y-intercept and the slope. Increasing churn, γ, has little to no effect on either the y-intercepts or the slopes of the lines in Figure 4.9. Finally, increasing $|D|$ causes only a small decrease in both the y-intercepts and the slopes of the lines in Figure 4.10. Therefore, one can see that while the in-network probability has the biggest negative impact on information availability, increasing the delegation chain depth limit can be a viable way to significantly increase information availability.

4.2.6 Delegation Strategies.

Consider a simple delegation strategy in which p_ℓ chooses a random set D, and any time $\pi_{re\,del}$ is run, a new random set D' is chosen. With each delegation (or redelegation), there is some chance that the delegated information will leak. This happens when all parties in the delegation set are corrupt. Let $|D|$ be the size of the delegation set, which, for simplicity, is assumed to be constant, but the protocols will work for different sized sets. Therefore, there are $\binom{c|P|}{|D|}$ sets of size $|D|$ for which all parties in the set are corrupt. (4.1) gives the probability of choosing a delegation set where all parties are corrupt, or in other words, the probability of a single delegation (or redelegation) resulting in leaking private reputation information given the delegation strategy just described.

$$prob\_leak = \frac{\binom{caN}{|D|}}{\binom{aN}{|D|}} \tag{4.1}$$

Using the parameters from the first simulation, (N=10000, $a = 0.75$, $c = 0.2$ and $|D|$=5), $prob\_leak \approx 0.0003$. Therefore, with 1700 delegations or redelegations total, the probability that the private reputation information would have leaked is $0.0003(1700) = 0.51$. With high churn rates in a network, one can expect a lot of delegations and redelegations and would have to stop delegating at some point in order to guarantee security. Therefore, a better delegation strategy is needed. This sections studies two delegation strategies. One provides strong privacy guarantees but could potentially leak some information about who p_ℓ trusts (but not the actual reputation values). The other has weaker privacy guarantees, but does not leak information about who p_ℓ trusts.

4.2.6.1 Guaranteed Privacy.

Since p_ℓ has reputation information on other trusted parties in the network to guarantee privacy, this information helps p_ℓ when choosing how delegation should work, i.e., the initial set D and the delegation chain depth δ. Let p_ℓ choose δ according to how available he wants his information to be when out of the network. For example, this could be determined based on the churn rate of the network. Once δ is set, p_ℓ forms the set D_h of parties that he trusts the most (based on reputation values he possesses) where the size of D_h is $\lceil \frac{\delta}{a} \rceil$. These parties are known by p_ℓ to be honest and will help provide strong privacy guarantees by forming part of D. p_ℓ also chooses some number of other parties from the network at random, whose trustworthiness is possibly unknown. Call this set D_u. When p_ℓ would like to leave the network, he runs protocol π_{del} with $D = \mathit{in\,network}(D_h \cup D_u)$, where $\mathit{in\,network}$ returns the subset of the parameter of those parties which are currently in the network. At a later point when a party, say $p'_\ell \in D$, wants to leave the network, the parties in D run $\pi_{re\,del}$ with $D' = D - \{p'_\ell\}$.

Due to the way D_h is constructed, $|\mathit{in\,network}(D_h)| \approx \delta$. Furthermore, there are approximately $(1-c)|D_u|$ honest parties in D_u. Therefore, at any moment in time, the set D will contain at least δ honest parties. Since δ limits the delegation chain, it is guaranteed that there will always be at least one honest party in the redelegation sets. Therefore, privacy is ensured.

4.2.6.2 Probabilistic Privacy.

One can make the delegation strategy simpler by relaxing the security guarantees. Let p_ℓ choose a set D_u at random of size $\frac{1-c}{a\delta}$ where δ is the desired delegation chain depth to ensure some level of availability. Set $D = \mathit{in\,network}(D_u)$ when p_ℓ runs π_{del}. When the parties in D run π_{redel}, they set $D' = D - \{p'_\ell\}$. Due to the way D_u is chosen, there should be δ honest parties in D at any instant in time, and since the delegation chain depth is limited by δ, there will always be at least one honest party in the redelegation sets. In practice, D_u should be somewhat larger in order to have even stronger assurances of privacy.

4.2.6.3 Discussion.

The first delegation strategy is able to provide better privacy guarantees by exploiting the reputation values that p_ℓ possesses. There are circumstances where this could leak information about who p_ℓ trusts but not the actual reputation values of p_ℓ. This may or may not be of concern, depending on the application. The second delegation strategy does not have this problem, but is not able to provide as strong of privacy guarantees, though this strategy could be very viable in networks where c, the fraction of corrupt parties, is very low. The description of the second strategy requires knowledge of c, which is a drawback, but conservative estimates of c can likely be computed.

4.2.7 Implementation.

This section presents an implementation of the four protocols presented earlier, in order to better understand the timing characteristics of the protocols. The implementation is in the Python language, and all communications take place using the Python remote object functionality provided by Pyro [70]. For the finite field for additive secret sharing, the implementation uses $\mathbb{Z}_{1021}$. This field is more than sufficient as the maximum reputation value is 10 and the query set sizes are small.

The primary component of the implementation is the *Agent*. An agent is a party in the network. Each agent begins with some amount of reputation information on other parties in the network. This bootstraps the reputation system. While not done in the experiments, an agent could start with no reputation information. Agents register with the Pyro nameserver to make their availability in the network known. They are then free to communicate with each other. For the purposes of the implementation, query and delegation sets are chosen randomly and one can set the size of each of these sets programmatically. Assume the same sizes across the whole network, though in practice they can differ. For th delegation strategy, the redelegation set is equal to the previous delegation set minus the party that is leaving. This sets a natural bound on the delegation chain to be the size of the original delegation set minus two. That way there will always be at least two parties in the delegation set. In practice, one would need to be more careful in choosing the delegation set and setting the bound on the delegation chain depth accordingly, as discussed in the previous section. For the purposes of the timing experiments, this delegation strategy will suffice.

4.2.8 Experimentation.

Using the implementation detailed in the previous section, this section reports on a number of experiments to demonstrate the run-time efficiencies of the Dyn-PDRS protocols. This section, describes the results. For the experiments, let $N = 50$, $a = 0.9$ when delegation is used, $s = 0.5$ and $\gamma = 0.1$ unless otherwise stated. An explanation of these symbols is given in Table 4.3.

Figure 4.11 shows the timing information for running π_{add} with various query set sizes and a fixed delegation set size ($|D| = 6$). Notice that the time to execute π_{add} increases as the query set size increases. This plot also reveals a lot about π_{act}. π_{act} is called as a subroutine of π_{add} when delegation is enabled and a party that appears in the query set has left the network. In which case, the parties in the delegation set act on his behalf. The plot also shows the effect of π_{act}, both the overall time to execute and the slope increase. Even with a query set size of ten, however, π_{add}, both with and without delegation is very fast. Figure 4.12 shows the results of a similar experiment but this time varied the size of the delegation set and fixed the size of the query set to five. With no delegation, the delegation set size has no effect. Again, notice how the time to execute π_{add} with delegation increases as the size of the delegation set increases.

Effect of $|U|$ on π_{add} execution time

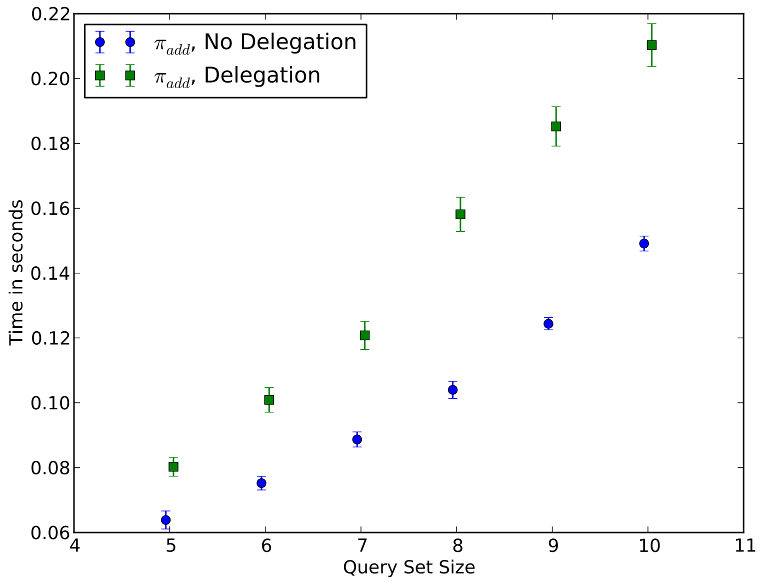


Figure 4.11: Average time to execute π_{add} with varying query set size and 95% confidence interval.

Figures 4.13 and 4.14 plot the average time (with 95% confidence interval) to run π_{del} and $\pi_{re\,del}$ respectively, with varying delegation set sizes. The query set size has no effect on the running time of these protocols and is fixed at five for these experiments. In Figure 4.13, notice that π_{del} is a very fast protocol and increases linearly as the delegation set size increases. Figure 4.14 reveals that $\pi_{re\,del}$ is the most expensive protocol in the Dyn-PDRS. With smaller delegation set sizes, however, it is still practical.

All of the previous plots showed the average execution time over the entire experiment. The time to execute π_{del} and $\pi_{re\,del}$ can vary greatly depending on how much information needs to be delegated or redelegated. To better understand how the amount of information affects the running time of these protocols, we plot the individual data points collected during an experiment in which s (the fraction of information bootstrapped into the reputation system) is varied and, upon either a delegation or redelegation, counted

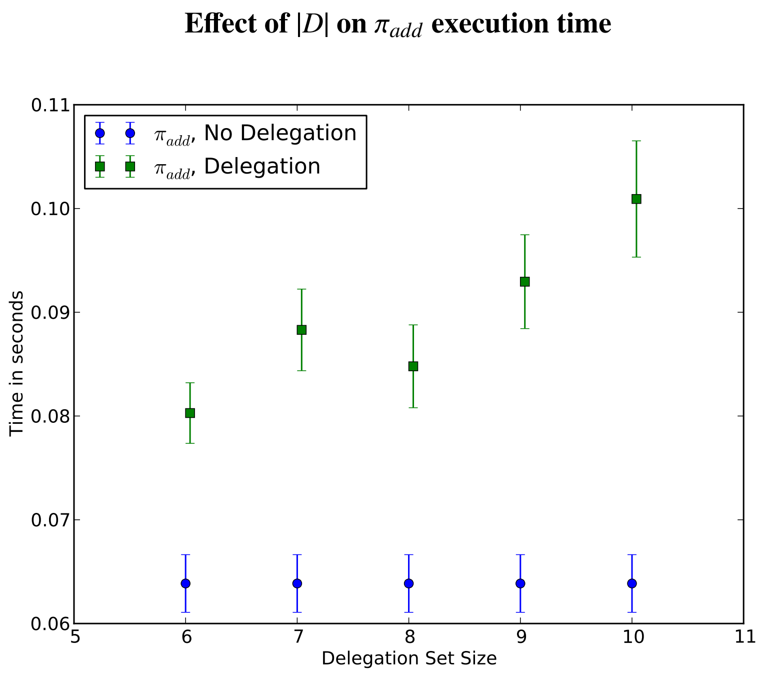


Figure 4.12: Average time to execute π_{add} with varying delegation set size and 95% confidence interval.

the number of reputation values being delegated or redelegated respectively. For this experiment, the query set size is fixed at five and the delegation set size at six. Figure 4.15 shows the results of this experiment for π_{del} and Figure 4.16 for $\pi_{re\,del}$. Each plot includes the linear least squares regression line. For both protocols, the execution time increases linearly as the amount of information increases, though the slope of the line is much higher for $\pi_{re\,del}$.

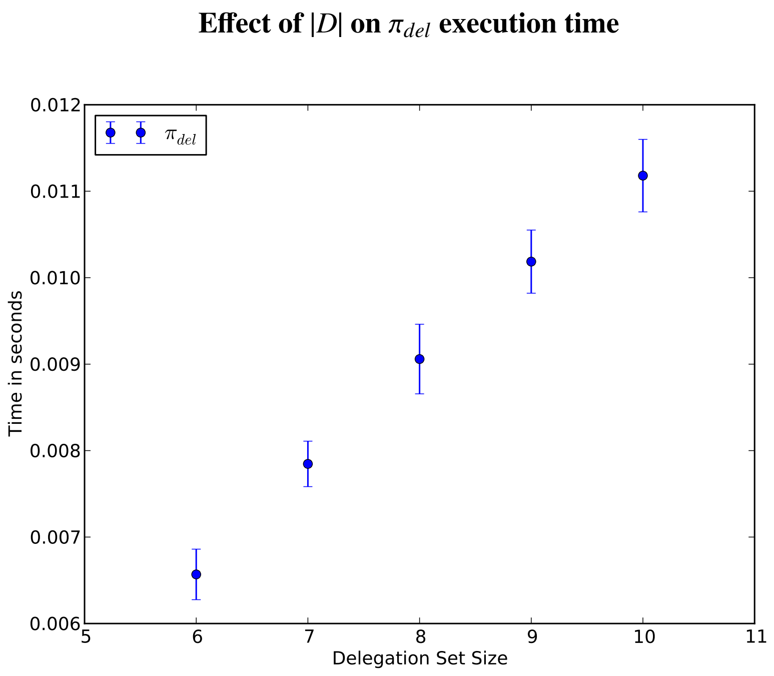


Figure 4.13: Average time to execute π_{del} with varying delegation set size and 95% confidence interval.

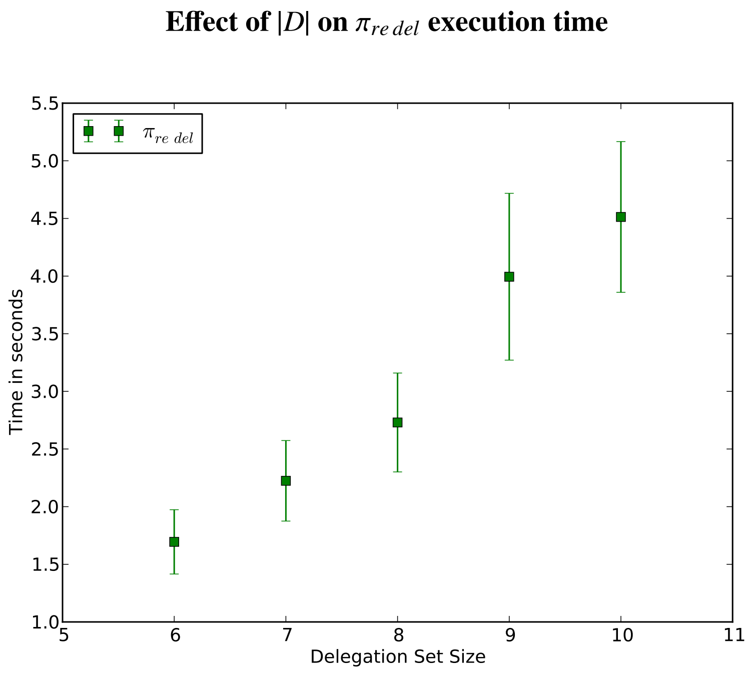


Figure 4.14: Average time to execute $\pi_{re\,del}$ with varying delegation set size and 95% confidence interval.

Timing plot for π_{del}

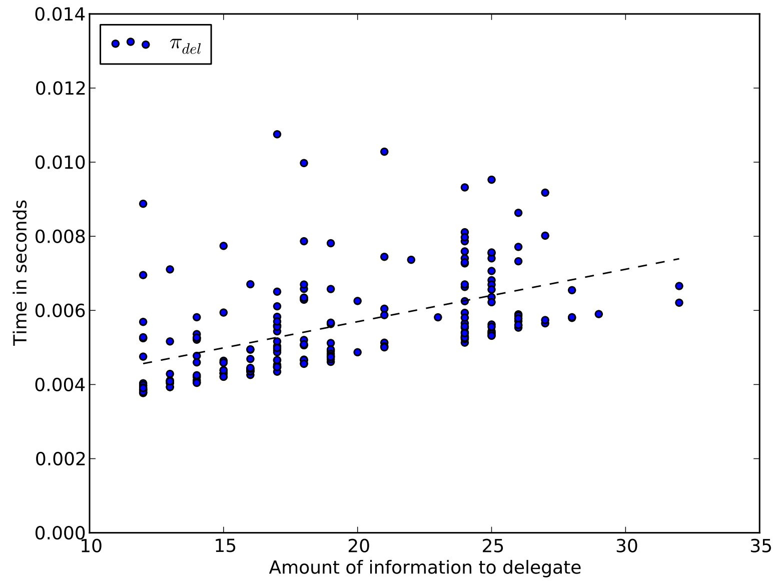


Figure 4.15: Timing as amount of information increases for π_{del}.

Timing plot for $\pi_{re\,del}$

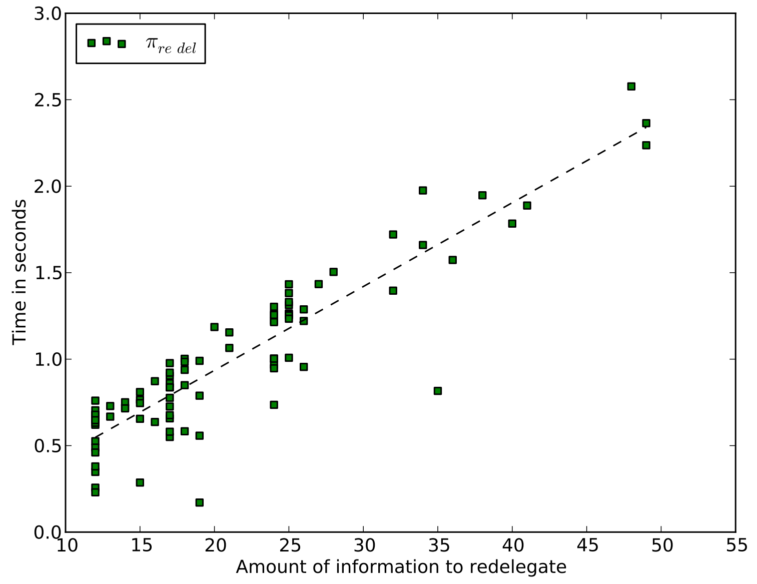


Figure 4.16: Timing as amount of information increases for $\pi_{re\,del}$.

V. Adversary Model Tradeoffs

Adversary modeling is a very important tool in the design of security protocols. It forces a protocol designer to specify exactly the conditions under which the protocol will be secure and often leads to a formal proof of security. Chapter 3 shows examples of adversary modeling and security proofs. The two most common adversary models are the honest-but-curious model (or semi-honest) and the malicious model. Each adversary model has its advantages and disadvantages. This chapter presents methods to study the tradeoffs between the two models. Protocols in the honest-but-curious model are often more lightweight than their malicious model counterparts. They typically have lower computation requirements, lower communication requirements, or both. The disadvantage of these protocols is that the honest-but-curious (HbC) assumptions might not be realistic, especially if the value of an attack is high. This is especially true in smart metering applications where the meters are geographically separated and thus, not under the physical protection of some entity with a vested interest in keeping them safe. That is why often HbC privacy preserving smart metering protocols have the additional assumption that devices tamper resistant. Malicious model protocols are inherently more resilient and do not require tamper resistant hardware, but often have a higher cost in computation or communication or both.

When designing a privacy-preserving system such as in the two applications described in Chapter 4, choosing a realistic adversary model given the context of the deployed hardware/software is very important. Previous research has not addressed how much less efficient malicious model protocols are, and what is the effect of anti-tamper on HbC protocols. Answering these questions is very important as it could potentially have a major impact on the efficiency and security of the system. The results in this chapter present analysis of protocols geared towards the smart grid application that use three different paradigms for privacy-preserving computation: homomorphic encryption, multiparty computation, and transferable MPC.

5.1 Motivation

Consider the problem of spatial aggregation of load data in a network of N meters and one node called the sink node. The sink node represents the party authorized to learn the final aggregated value. Erkin and Tsudik [55] present a protocol that is secure in the honest-but-curious model which requires each meter to perform one encryption, one hash function computation and generate $N - 1$ random numbers. The aggregation node then homomorphically aggregates the information from the N meters and decrypts the result. Garcia and Jacobs [56], on the other hand, present a protocol that is secure in the malicious model which requires $N - 1$ encryptions and one decryption per meter. Furthermore, the aggregation node must homomorphically aggregate approximately N^2 different values before getting the final result. Clearly the malicious model protocol requires much more computation for both the aggregation node (N^2 aggregations vs. N) and for the meter nodes (public-key encryptions vs. random number generation).

As illustrated by the previous example, there are many tradeoffs between the two adversary models. This section presents methods for understanding these tradeoffs in two different ways. The initial discussion focuses on existing smart meter aggregation protocols that use homomorphic encryption from the literature. Their respective communication and computation requirements when applied to sample smart meter hardware is presented. Next, the discussion turns to generic secure multiparty computation protocols that can compute almost any function on private inputs. Such protocols have only recently been studied for their application to the smart grid. These protocols provide an interesting avenue to help us further understand the implications of the different adversary models when developing privacy protocols for the smart grid. Protocols for each adversary model are studied to understand their requirements and compare them to understand better adversary model tradeoffs. Furthermore, while a complete, in-depth study of smart meter anti-tampering is outside the scope of this paper, Section 5.6 presents information on various anti-tamper protections that are likely candidates to be used in future smart meters and discuss the costs associated with each. The intent here is not to say categorically which adversary model is the best, but instead the analysis will give an understanding of the issues to consider when choosing one adversary model over the other and present methods for conducting these analyses.

Privacy-preserving protocols based on homomorphic encryption have very high costs associated with moving from HbC to malicious model protocols. MPC based protocols are richer since they allow more complex computations. The information in this chapter presents an understanding of the adversary model tradespace in smart metering systems. The computations studied are privately computing sums of consumption information across a neighborhood or city and standard deviation of consumption information across a neighborhood or city. This chapter proposes metrics for comparing fundamentally similar protocols from different adversary models.

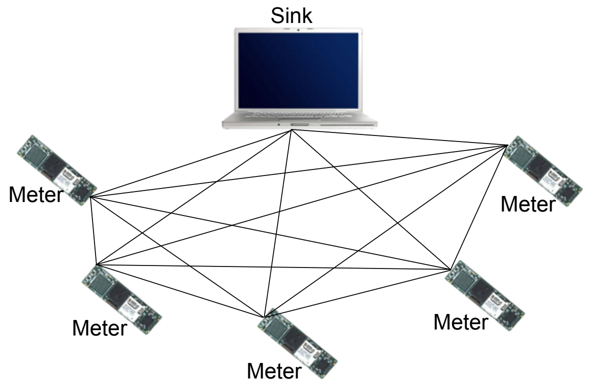


Figure 5.1: Fully Connected Network Model.

For the protocols of interest to operate properly, any network topology will work as long as every node can talk to every other node. This can be accomplished via routing or direct connections. For simplicity, assume a fully connected topology as shown in Figure 5.1. The number of meter nodes may vary, but there is always a single sink. Meter nodes use the Gumstix Overo Earth with a 600MHz processor and 256MB of RAM. The sink node, is a computer running an Intel Core i5-540M CPU and 4GB of RAM.

5.2 Homomorphic Encryption in Smart Metering

Probably the most common computation found in the literature for privacy preservation in smart grids is aggregation of usage measurements over a spatially separated area (e.g., a neighborhood or city). A number of proposed protocols attempt to solve this problem. This study focuses on two, a protocol due to Erkin and Tsudik, called the ET protocol [55], and a protocol due to Garcia and Jacobs, called the GJ protocol [56]. These specific protocols have been chosen for the following reasons: (1) they are built using homomorphic encryption; (2) they are fairly similar in functionality, yet the ET protocol is secure in the honest-but-curious model while the GJ protocol is secure in the malicious model; and (3) they fit the network model well. By restricting this section to protocols that use homomorphic encryption, recent works involving differential privacy (e.g., [71]) are not considered. Other protocols initially considered include work by Li et al. [59] which is secure in the HbC model but assumes a different network structure from ours and work by Shi et al. [58] which appears to be secure in the malicious model (though no adversary model is claimed or proven in the paper) but uses a different cipher (a modified version of ElGamal). By using two protocols that use the same cipher and the same network structure, effects of the different adversary models are best revealed.

The remainder of this section presents an overview of each of these protocols and then presents timing measurements for the basic operations necessary to carry out the protocols as measured on the devices in the example network.

5.2.1 ET Protocol.

In [55], Erkin and Tsudik present protocols for spatial, temporal and spatio-temporal aggregation in smart meter networks, each secure in the honest-but-curious model. This section reviews only the spatial aggregation protocol as it will be the only one used in the experimentation and analysis.

The ET protocol uses a slightly modified version of the Paillier cryptosystem. The modification is that each ciphertext has a noise component added to it. The noise components are generated among the meters in such a way that when all the ciphertexts are homomorphically aggregated, the noise disappears. This allows for correct decryption of the final aggregated value while making decryption of an individual ciphertext impossible, thus preserving the privacy of the individual meters. Each meter knows the public key of the sink node. This is a slight simplification of the protocol in the original paper but is sufficient for the analysis in this chapter. The following steps outline the protocol to aggregate the usage data for all meters at one instance in time:

1. Each meter generates $N-1$ random values and sends one random value to every other meter in the network.
2. Each meter uses the $N-1$ random values it received plus the $N-1$ random values it sent to come up with the noise component for encryption.
3. Each meter encrypts the current usage information using the sink's public key and the noise component calculated in the previous step.
4. The meters send their respective encrypted information to the sink.
5. The sink takes all N encrypted values, homomorphically aggregates the values and decrypts to obtain the final result.

5.2.2 GJ Protocol.

The malicious model protocol used in the analysis below comes from Garcia and Jacobs [56]. Like the ET protocol, it is a protocol used for spatial aggregation of usage data preserves privacy. The authors never explicitly state which cryptosystem to use but mention Paillier as a possibility, which is used in this chapter. Each meter has a public-private key pair associated with it, and the meters know each other's public keys. The GJ protocol works as follows:

1. Meter i takes its current usage reading, m_i, and splits it into N shares $(a_{i1}, \cdots, a_{iN})$ such that $m_i = \sum_{j=1}^{N} a_{ij} \bmod n$ (where n is a large number known to all meters).
2. Meter i then encrypts each share using a different meter's public key, $d_{ij} = E(a_{ij}, PK_j)$, except for its own share a_{ii}.
3. The meters send the encrypted shares to the sink node.
4. The sink node homomorphically aggregates the shares encrypted with the same public key. In other words, for PK_i the sink aggregates $d_{1i}, d_{2i}, \cdots, d_{Ni}$ and sends the aggregated value to meter i.
5. Meter i then decrypts the aggregated value from the previous step, adds in a_{ii} and returns the result to the sink node.
6. The sink node adds up all the values received from the meters which is shown to equal the aggregate sum of the usage information.

5.2.3 Timing Measurements.

	Encrypt	Decrypt	Aggregate	GenRnd
Meter	929ms	903ms	7.3ms	0.15ms
Sink	98ms	97ms	0.47ms	N/A

Table 5.1: Timing for homomorphic encryption operations.

This section presents results on the computation and communication requirements for all of the necessary operations in the ET and GJ protocols. These include the time to encrypt and decrypt using Paillier, to homomorphically add two ciphertexts, to transmit and receive the various message types, etc. Section 5.5 uses this information to extrapolate the communication and computation requirements of the full protocols. The Paillier implementation used for the measurements is the thep library (http://thep.googlecode.com/) which is written in Java but configured to perform the most expensive large integer operations using GMP (http://gmplib.org/) for better performance. The keys are 2048-bits and the timing information comes from the over 500 runs for each operation. The results are shown in Table 5.1.

For communications, assume that each node in the network has a wireless link capable of 250kbps throughput. Furthermore, assume that processing, propagation and queuing delays are negligible and that packet headers are negligible in size and do not contribute significantly to the size of the overall packet. These assumptions are justified due to the simplicity of the network. Therefore, the only delay under consideration is transmission delay. Using this information, the estimated time to transmit a 128bit random number is 0.512 milliseconds and a 2048bit ciphertext takes 8.192 milliseconds.

5.3 Multiparty Computation in Smart Metering

Section 2.3 presents a brief introduction to secure multiparty computation (MPC). MPC could prove to be an important tool in the smart grid as it would allow meters to perform much more complex computations in the grid without compromising the privacy of the individual parties, but there has been little research in applying MPC to the smart grid to date. Danezis, et al. [72] present a protocol for privacy preserving billing and mention smart metering as one potential application. Their protocols use many of the building blocks of MPC and are therefore highly related. Thoma, et al. [73] apply multiparty computation techniques to the smart grid, but only look at secure summation and secure comparison. Using these two functionalities, the authors propose a system which provides demand management and billing with verification. They do not look, however, at generic multiparty computation which can compute any function. Furthermore, their work only looks at the honest-but-curious adversary model and does not look at tradeoffs between the two models. Peter et al. [74] propose using MPC for smart grid computation in a slightly different model than the one considered here. This work builds privacy-preserving computations that occur in-network. In their work, Peter et al. look at using MPC in the outsourcing model where they assume that the meters have access to computation servers (e.g., the cloud) which perform the computation privately, on behalf of the meters.

In contrast to previous work, generic multiparty computation protocols can compute advanced functionalities such as standard deviation, statistical hypothesis tests, etc. Benchmarks for MPC are measured using the VIFF (http://viff.dk) framework. VIFF provides protocols in both the honest-but-curious and the malicious models. This section introduces VIFF and presents overviews of both of the protocols of interest. The section also presents timing measurements for the critical components of each protocol that are used later use to understand the tradeoffs in adversary models.

5.3.1 Introduction to VIFF.

This section presents a brief overview of VIFF. For an in-depth guide to VIFF, see [47]. VIFF is written in Python and allows a developer to specify multiparty computations using a simple API and a runtime environment that handles all the complex operations necessary to carry out the MPC. VIFF includes runtimes which are secure in both the honest-but-curious model as well as the malicious model and assumes an asynchronous network.

When using VIFF, the four main commands are **input**, **output**, **multiply**, and **add**. **input** allows a party to enter their input to the computation. It uses secret sharing to securely split an input s into $s_1, s_2, \ldots, s_n$ such that any t of those shares can be combined to recover the original value. The shares are then distributed to the other parties. **output** allows the authorized parties to learn the value that corresponds to a previously secret shared value. In other words, when **output** is called on s, then the parties would reveal their shares of s to the authorized parties. **add** allows us to add together two secret shared values such that each party has a share of the result. For example, to compute $s = a+b$, each party should end up with a share of s (say s_i) without the values of a, b, or s being revealed. Similarly, **multiply** results in each party holding a share of s such that $s = ab$ without revealing any of the values. Given these four commands, any arithmetic circuit can be computed. To compute f, represent it as an arithmetic circuit built up of add and multiply gates. Secret share the inputs among the parties and use **add** when at each add gate and **multiply** for each multiplication gate. When the circuit is fully executed, use **output** on the final gate(s) of the circuit. VIFF runtimes implement the necessary protocols to carry out these four commands. The specific runtimes of interest are *viff.passive.PassiveRuntime* and *viff.active.ActiveRuntime*. The *PassiveRuntime* is secure in the honest-but-curious model for $t < n/2$ corrupted parties. The *ActiveRuntime* is secure in the malicious model for $t < n/3$ corrupted parties. The subsequent sections, present details on how these protocols work as they form the basis of the analysis presented in this chapter.

5.3.2 *Timing Measurements (honest-but-curious).*

Table 5.2: Coefficients for timing estimation polynomial of HbC model operations ($y = c_2x + c_1$ with y and c_1 in milliseconds, c_2 in milliseconds/party, x is number of parties).

	input		**output**		**add**	**multiply**	
	c_2	c_1	c_2	c_1	c_1	c_2	c_1
Meter	1.178	−1.019	0.056	0.289	0.07	1.234	−0.66
Sink	0.043	−0.036	0.003	0.01	0.002	0.049	−0.026

To understand the communication and computation requirements of *PassiveRuntime*, which leads to the timing measurements for the runtime, consider how the four commands listed in Section 5.3.1 operate. Table 5.2 shows timing measurements for the commands (not including communication time). For the **add** operation, the table gives the time to add two numbers. The other commands are not as simple. Note, however, that the remaining three are of linear complexity (in the number of parties), so a polynomial fit ($c_2 x + c_1$) can be used. The table gives the coefficients of the polynomial.

The **input** command uses Shamir secret sharing (SSS) [42] to split the input s into n different shares ($s_1, \ldots, s_n$) such that any $t + 1$ of the shares can be used to recover s. To do this, party p_i chooses a random polynomial $\sigma(x) = s + r_1 x + r_2 x^2 + \cdots + r_t x^t$ where s is the secret to be shared and each r_i is a random value from some finite field. p_i then computes the shares $s_j = \sigma(j)$ for j from 1 to n and sends s_j to party p_j and keeps one share for himself. With fewer than $t + 1$ shares it is not only impossible to recover s but additionally no information about s is leaked. The communication cost of this command is the cost of communicating the n shares. The computation cost is simply the cost of generating random coefficients then evaluating the polynomial n times. **output** is really the reverse of **input**. At least t shares must be communicated to the parties who are allowed to learn the output. Those parties then use the shares to reconstruct the output value. If k parties are allowed to learn the output, the communication cost is that of communicating kn shares. The computation cost is that of reconstructing the shares. SSS uses Lagrangian interpolation for reconstruction, which is quite fast. Given at least $t+1$ shares reconstruction of s is computed by the following equation.

$$b_i = \prod_{j \neq i} \frac{j}{j - i} \text{ and then } s = \sum_i b_i s_i$$

The **add** command is very simple. To compute $f = d + e$, where each party i holds shares d_i and e_i, due to the nature of SSS, they simply compute $f_i = d_i + e_i$. The shares f_i are proper shares of $f = d + e$. Therefore, the computation requirements of **add** are very small and there is no communication required. **multiply**, on the other hand, is a more expensive operation. To compute shares of $f = de$, each party first computes $f'_i = d_i e_i$, then secret shares f'_i so that party j has f'_{ij}. Each party j then uses the shares $f'_{1j}, f'_{2j}, \ldots, f'_{nj}$ to reconstruct f_j, which is a proper share of f. Thus the communication cost is that of communicating the shares and the computation cost is that of generating n shares and reconstructing with n shares. This protocol is secure in the HbC model [75].

5.3.3 Timing Measurements (malicious).

The VIFF malicious model protocol is presented in [50]. Table 5.3 presents the timing measurements for the protocol. The table omits **add** and **output** as, for the sizes of parties considered here, they are roughly the same as in the passive runtime. When working in the malicious model, a new command (**genTriple**) is used. Furthermore, the commands are used in two phases, preprocessing and computation. The **genTriple** has quadratic complexity (in the number of parties) so the table presents the coefficients of the equation $y = c_3x^2 + c_2x + c_1$ in the table.

Table 5.3: Coefficients for timing estimation polynomial of malicious model operations ($y = c_3x^2 + c_2x + c_1$ with y and c_1 in milliseconds, c_2 in milliseconds/party, c_3 in milliseconds/party squared, x is number of parties).

	genTriple			**input**		**multiply**	
	c_3	c_2	c_1	c_2	c_1	c_2	c_1
Meter	0.141	4.879	−3.209	1.178	−1.019	0.112	0.578
Sink	0.002	0.181	−0.114	0.043	−0.036	0.006	0.02

The preprocessing phase handles **genTriple** and **input**. **genTriple** generates shares of multiplication triples, i.e., values d, e, f such that $f = de$, for each party. Such multiplication triples can be generated using hyperinvertible matrices [76] or pseudorandom secret sharing [77] with the latter being more efficient for small values of n [50], thus the timing measurements here use the hyperinvertible matrices method. **input** is also slightly modified from the passive runtime. Instead of secret sharing the input, the protocol instead secret shares a random value r and broadcast the value plus the input if preprocessing terminates successfully.

The computation phase is where the other commands take place. **add** and **output** are carried out exactly as in the passive runtime. **mult**, however, is different. For each multiplication that must be performed, each party uses the shares of a multiplication triple from the preprocessing phase, say d_i, e_i, f_i. Furthermore, each party has shares of the values to be multiplied, say a_i, b_i. The result of **mult** should be a share of $g = ab$ for each party. Let $d'_i = a_i - d_i$ and $e'_i = b_i - e_i$. The parties then publicly reconstruct d', e' using **output**. Each party then computes $g_i = d'e' + d'e_i + d_ie' + f_i$ which is their share of $g = ab$. Thus, the communication and computation requirements of **mult** are almost entirely based on the requirements for **output** which is called twice.

5.4 Transferable Multiparty Computation in Smart Metering

In Chapter 3 presents the honest-but-curious and malicious model protocols for transferable multiparty computation. Recall that T-MPC builds upon existing MPC protocols, like those listed in the previous section, by adding two additional functions, **transfer** and **recombine transfer**. These functions can be used to privately transfer computations between sets of parties. Section 4.1 describes how these protocols have been applied to the smart grid to enable much more efficient computations with much higher scalability.

5.4.1 Timing Measurements (honest-but-curious).

The timing information for HbC T-MPC is computed in a similar manner as what was done previously for MPC. In fact, the **input**, **output**, **add**, and **multiply** functions, are the same, so the same timing measurements can be used. The function **transfer** is called by all parties in one set in order to transfer an intermediate value to a new set of parties. The parties call **transfer** with their share of the intermediate value as input. **transfer** creates subshares of the share by using Shamir secret sharing with the threshold set to the size of the new set of parties divided by two. Each party in the original set then sends one subshare of their intermediate value to each party in the new set.

Upon receiving all the subshares, each party in the new set calls **recombine transfer** on those subshares. **recombine transfer** runs Lagrangian interpolation on these subshares to get a new share. Thanks to the linear nature of SSS, the new shares held by the parties in the new set form shares of the intermediate value transferred by the parties in the old set. If there are multiple intermediate values that need to be transferred, these two protocols can be called multiple times.

5.4.2 Timing Measurements (malicious).

T-MPC in the malicious model uses the malicious model MPC protocols from above. For the **transfer** and **recombine transfer** protocols, recall that a result due to McEliece and Sarwate makes reconstruction robust [51]. They noted that Shamir secret sharing is basically a special form of Reed-Solomon codes. In particular, if the dealer is trusted and the fraction of corrupt parties is less than one third, then a Reed-Solomon decoder can be used instead of Lagrangian interpolation for reconstruction. In this case, reconstruction is "robust", i.e., guaranteed to return the correct value. Using this result, the **transfer** protocol for the malicious model is the same as what was used in the HbC case. For **recombine transfer**, Reed-Solomon decoding replaces Lagrangian interpolation. Specifically, since the size of the subgroups considered here are will be fairly small, a brute force Reed-Solomon decoder is sufficiently fast. The malicious adversary model for (T-)MPC, assumes a threshold of $n/3$ for the number of corrupt parties. The result of McEliece and Sarwate described above requires an honest dealer. For T-MPC, however, this is not necessary. To see that this is not an issue, consider where the subshares come from. Each party in the new set receives one subshare from each party in the old set. Therefore, there is no single dealer of the subshares. The "trusted dealer" requirement is taken care of by the fact that collectively, the old set of parties acts as a trusted dealer. Specifically, each party in the new set is guaranteed to receive no more than $n/3$ corrupt subshares. For timing **recombine transfer**, simply multiply the timing information for **output**, from above, by the number of times required to guarantee robust reconstruction.

5.5 Analysis

5.5.1 Homomorphic encryption based aggregation protocols.

Using the descriptions of the aggregation protocols from Section 5.2 one can determine the time required to run the protocol for any number of nodes in the network. This is useful in understanding the tradeoffs in the adversary models as it gives a real world sense as to the size of the network that each protocol can support. Note that overall asymptotic complexities will not capture this information as, for example, both aggregation protocols studied here have quadratic complexities overall (communication plus computation). Figure 5.2 shows the results of this experiment.

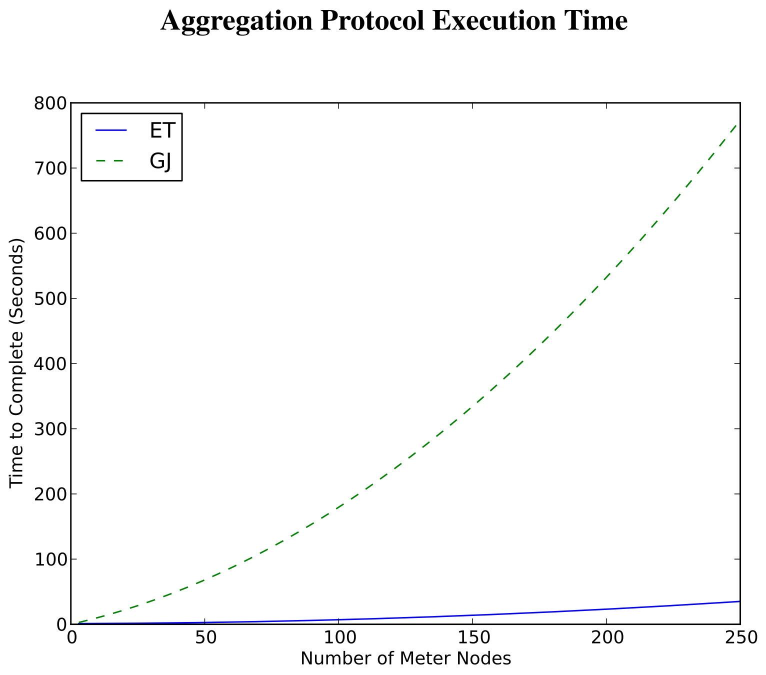


Figure 5.2: Size vs. execution time for aggregation protocols.

As expected the time to execute the GJ protocol is significantly higher than the ET protocol. What is really interesting about this graph, however, is that it gives us an idea of the size of network or the granularity of usage information that can be achieved using the more expensive yet more secure GJ protocol. For example, if a sink desires usage information every five minutes, the network size can be up to around 140 meter nodes. On the other hand, if a sink has networks of around 50 nodes, the protocol can be run as often as every 60 seconds and still use the malicious model protocol. Of further interest is which portion of the protocol contributes most to the time to complete protocol execution. For the ET protocol, once the system size exceeds 43 metering nodes, communicating the random values contributes most to the execution time. Prior to that, encryption contributes the most. For the GJ protocol, encryption contributes the most to execution time up until the network reaches 114 nodes, at which point communicating the meter value shares contributes most to the execution time. Therefore, for large smart meter networks, increasing communication bandwidth will have the biggest effect on execution time for both the ET and GJ protocols.

5.5.2 MPC Protocols.

In order to understand better the application of MPC protocols (and their corresponding adversary models) to the smart grid, the analysis in this section looks at computing the sum, standard deviation, and neighborhood standard deviation. The sum computed is identical to the sum that the aggregation protocols ET and GJ compute and therefore makes an interesting comparison with those works. When computing the standard deviation, the input for the meters is their current reading while the input for the sink is the current mean. This avoids computing a division in the multiparty computation which is a very expensive operation. Furthermore, the typical standard deviation equation involves a square root and a division, but to avoid the square root and the division, only the numerator is computed reveald to the sink who can compute the rest. For the regular standard deviation, the values x_i are the individual meter's readings for the neighborhood standard deviation, the x_i values in the equation are the sum of all meters in a neighborhood. To simplify the analysis, let each neighborhood contain 100 meters. The neighborhood standard deviation would be important to know in the smart grid as a high standard deviation would mean a lot of variability across neighborhoods, which could identify prime targets for more optimized generation or distribution methods, or could indicate the need for more detailed analysis.

Figure 5.3 plots the time to complete a protocol execution for each of the computations and for each of the adversary models. Furthermore, Table 5.4 shows the maximum number of meters each computation can support in under fifteen minutes. Interestingly, going from honest-but-curious to malicious when only computing additions almost comes for free. Additionally, using MPC to compute the sum is faster than using homomorphic encryption. Prior work on cryptographic methods for privately computing sums in the smart grid has often focused on additive homomorphic ciphers. Yet this work shows that in fact MPC would be a faster alternative. As seen with the aggregation protocols, moving from the honest-but-curious model to the malicious model comes with a fairly significant performance cost. As evidenced by the neighborhood standard deviation computation, however, if the number of multiplications can be limited, interesting functions on fairly large networks are possible.

Table 5.4: Maximum number of meters for less than 15 minutes of computation time.

	Sum	Std Dev	Std Dev (Neighborhood)
Honest-but-Curious	2647	777	2512
Malicious	2646	172	756

5.5.3 *T-MPC Protocols.*

Figure 5.4 plots the results of the experiment comparing HbC T-MPC and malicious model T-MPC for both the sum function and the standard deviation. As expected, executing the computation using a malicious model protocol is much more expensive. However, the network sizes in this case are still very large, illustrating the efficiency of T-MPC. From the figure, notice how the growth of the HbC executions and the malicious model executions

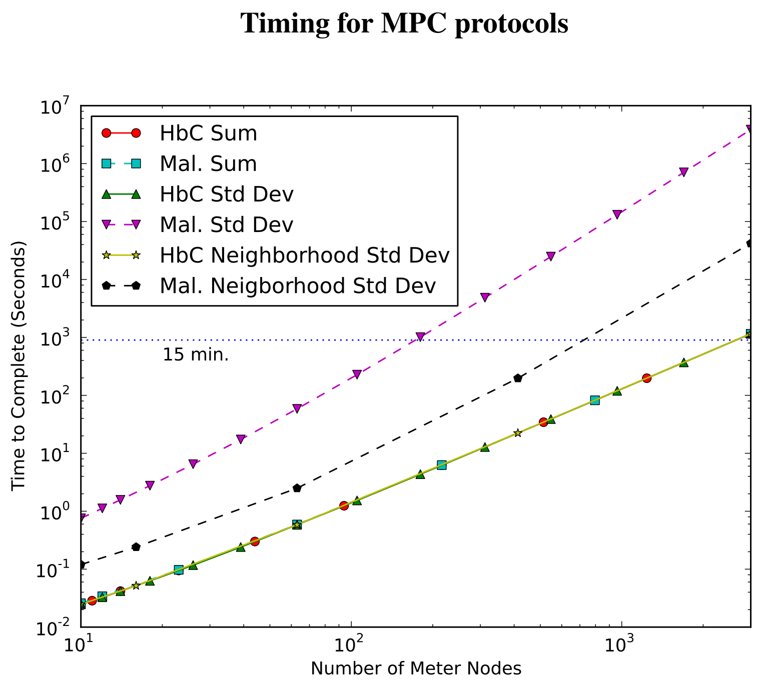


Figure 5.3: Network size vs execution time for MPC protocols.

are very similar. This means that the expected overhead of going from an HbC protocol to a malicious protocol is independent of the network size. Note that this is the case for both sum and standard deviation.

5.5.4 HE vs MPC vs T-MPC.

To finish the analysis by comparing the three types of protocols of interest, homomorphic encryption, multiparty computation, and transferable multiparty computation. The metric for comparing these protocols is the percent overhead when moving from an HbC protocol to a malicious model protocol. This will, in essence, tell us the cost of moving to a stronger adversary model and will have a significant impact on whether or not it makes sense to use an HbC protocol with anti-tamper protections or to simply use a malicious model protocol in the first place. Protocols that minimize this metric would be nice as then system designers could use malicious model protocols in the first place, simplifying system design from an anti-tamper prospective.

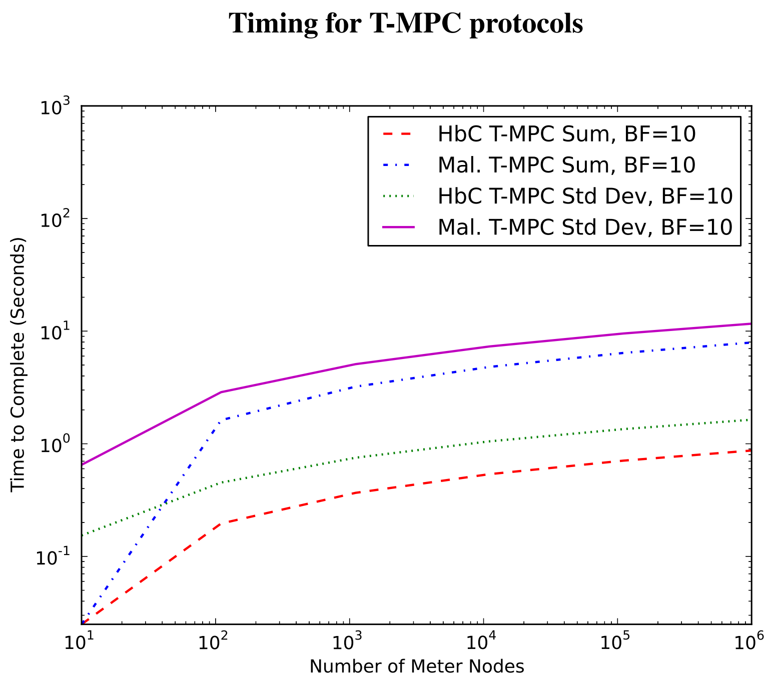


Figure 5.4: Network size vs execution time for T-MPC protocols.

Begin with the summation function and plot the percent overhead for increasingly larger networks in Figure 5.5. As before, MPC has practically no overhead when moving to a malicious model protocol when computing sums. T-MPC maxes out around 750% overhead for summation. While this is a significant increase, recall that T-MPC protocols are still the fastest in either model. Executing the summation function via T-MPC is still order of magnitude faster than MPC. Homomorphic encryption protocols have the highest overhead, maxing out around 2500%, but then going back down to just over 1500%.

Figure 5.6 shows a similar plot for the standard deviation function. This plot does not include homomorphic encryption in the comparison as the Paillier cipher is not able to compute that function. The comparison here is stark. T-MPC has roughly a constant overhead, i.e., independent of network size, and comes in at well below 1000%. MPC, however, does not share this property. The overhead continues to increase with network size. The fact that the overhead in moving from HbC T-MPC to malicious T-MPC is

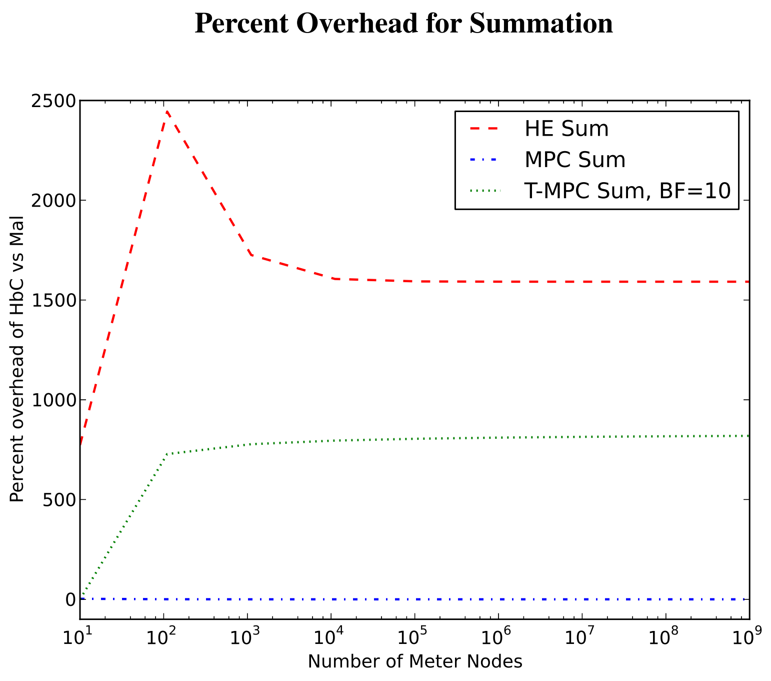


Figure 5.5: Percent overhead for HbC vs Malicious when computing summation.

independent of network size is very nice when trying to make a decision between HbC with anti-tamper and malicious model protocols. Network growth is not a concern as the overhead remains constant for ever larger networks. Combine this with the fact that T-MPC provided the fastest execution times, and there is good evidence to suggest that T-MPC provides the best balance in tradeoffs.

5.6 Anti-Tamper Protection

Existing smart meters provide very little in the way of security when it comes from physical attacks, reverse engineering, password extraction, eavesdropping and meter spoofing [78]. The threat of attacks has to make one wonder if protocols secure in the honest-but-curious adversary model are sufficient for real world deployment as an attack on these protocols that violates the adversary model could render the security mechanisms useless. Anti-tamper is an oft-cited way to make honest-but-curious protocols more

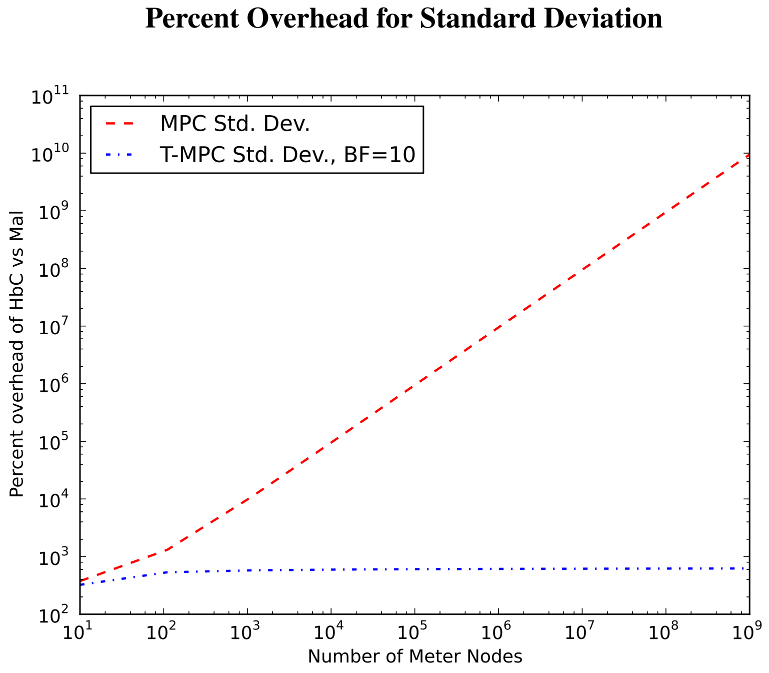


Figure 5.6: Percent overhead for HbC vs Malicious when computing standard deviation.

realistic in smart metering systems. Both hardware and software anti-tamper technologies exist which could provide meters which implement honest-but-curious protocols with the necessary protections to prevent violations of another's privacy. In fact, one would expect some combination of each to be present in future tamper resistant smart meters to provide maximum protection. Designing an anti-tamper solution for smart meters is outside the scope of this paper. Instead, this section surveys the existing literature and discusses costs associated with existing techniques. Note that it is possible that one could build honest-but-curious smart metering protocols for, say, aggregation that lend themselves well to AT protections to minimize the cost of adding those protections. This, however, is outside the scope of this work. Therefore, this section focuses on retrofitting existing protocols with AT protections in a very generic fashion. This section looks at AT protections that are available in the literature, note their performance costs, assume that a smart meter AT protection strategy would likely use a few of these protections. Generic AT factors represent using a few of these protections serially.

Work on anti-tamper protections applied to smart meters is scarce and presents an interesting problem. While there is significant room for improvement in developing a custom anti-tamper solution for smart meter devices, this is an exercise left to future work. McLaughlin et al. [79] propose a software technique for smart meters to promote firmware diversity that enhances a meter's ability to thwart compromise. Their system adds extra cycles to computation in order to encrypt and decrypt addresses during execution. The authors argue that since smart meter workloads are primarily I/O intensive that these protections should not decrease computation performance significantly. That said, using privacy preserving protocols on the meters would greatly increase a meter's computing. That said, their work does not deal with physical tampering of either hardware or software, something which would be necessary to thwart attacks on privacy.

The most common hardware approach to increasing a device's tamper resistance is to use a trusted processor [80]. These tamper-resistant processors can perform enough functionality to verify a system's components and software at boot-up or potentially during operation. The cost of these devices can range from the tens of dollars to the thousands, but smart cards are becoming a cheap, viable alternative. In addition to increased hardware costs, integration can also be an expensive cost up front. Real-time intrusion monitoring is also a potential avenue for anti-tamper in smart meters. In fact, some newer chips designed for smart metering have this functionality already built in [81]. On board sensors can check for physical intrusions and, when detected, can trigger other protection mechanisms (e.g., erase memory, warn the sink, etc). This would require a battery, as the mechanism would have to function even if power is lost. For large deployments, maintenance costs could be significant even at low false positive rates.

Software based anti-tamper protections require no additional hardware and can easily be changed/updated as needed. The problem with these sorts of protections is that they add computation overhead and therefore would slow down execution. One such technique, software encryption, aims to prevent an attacker from easily reverse engineering or tampering with the software by encrypting and only decrypting a certain function as needed. This of course increased computation time as decryption must occur. In one example system, computation time increased by up to a factor of eight [82]. Code obfuscation is another technique that attempts to make code difficult to analyze. If an attacker cannot analyze the code easily, they also cannot maliciously modify its functionality. This can be achieved by using a sequence of instructions which has the same effect as the original instruction. Furthermore adding instructions which will have no effect on the correctness of the computations performed but are instead aimed to confuse a reverse engineer, is another method of code obfuscation. One such system reported a slowdown factor as high as five [83].

To help illustrate how anti-tamper protections contribute to the discussion of tradeoffs between adversary models, in Figures 5.7, 5.8 and 5.9 plot the timing values of the protocols identified previously and include hypothetical anti-tamper computation factors. As seen previously, individual AT protections can result in a 5 to 8x performance hit. Running say two to three of these serially could easily lead to a 10 to 20x performance hit for the entire system. Figure 5.7 shows that with an anti-tamper (AT) factor of 20x, the timing of the ET protocol approaches that of the GJ protocol. For MPC, the contrast is even more stark. For all three computations, a 10x anti-tamper factor makes the honest-but-curious protocols less efficient than the malicious model counterparts for up to some number of meter nodes. This shows that under certain conditions, MPC honest-but-curious protocols operating under tamper protection can lose their benefit of being less cumbersome protocols. This is

Homomorphic Encryption with Anti-Tamper

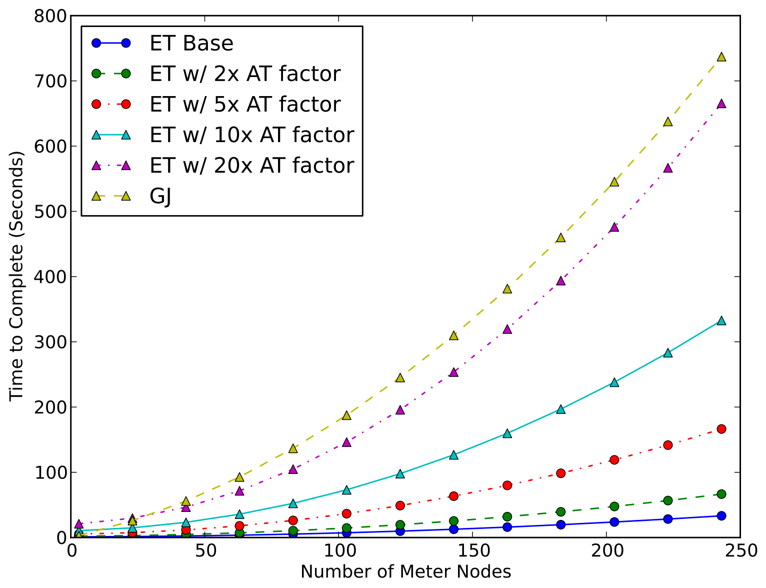


Figure 5.7: Homomorphic Encryption aggregation with various Anti-Tamper (AT) factors.

further illustrated when looking at the T-MPC protocols as shown in Figure 5.9. In both the summation and standard deviation functions, a 10x AT factor makes the HbC protocol less efficient than the malicious model protocol.

The analysis shows that care must be taken when choosing privacy preserving protocols for smart meter networks and consider the entire cost of that choice. Previous work often justified the use of the honest-but-curious adversary model by assuming anti-tamper protections could be added to make the security guarantees better fit the real-world threat model. Under certain assumptions about anti-tamper protections, malicious model protocols may be more efficient in the first place, potentially eliminating the need for anti-tamper protections entirely.

MPC with Anti-Tamper

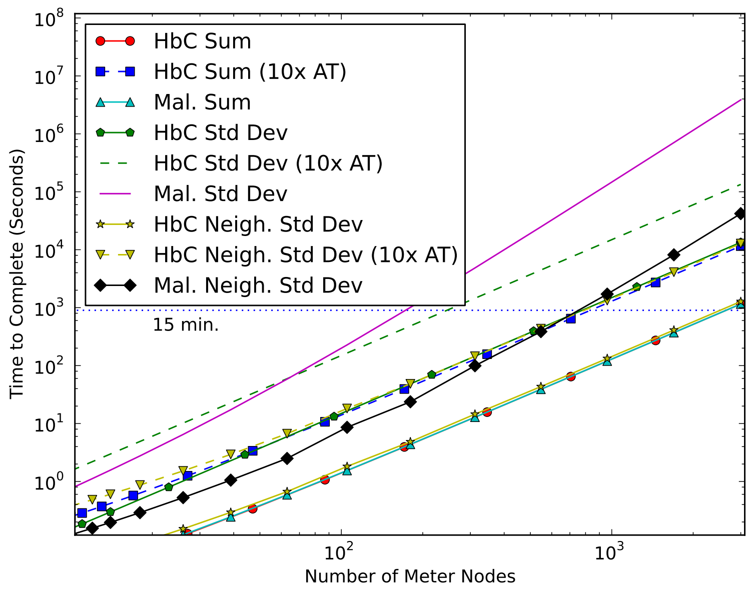


Figure 5.8: Multiparty Computation (MPC) timing with 10x AT factor.

T-MPC with Anti-Tamper

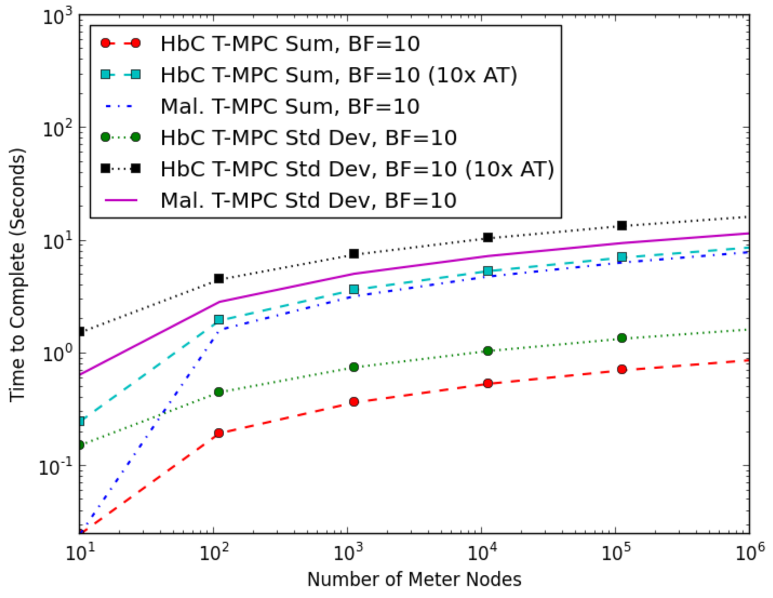


Figure 5.9: Transferable Multiparty Computation (T-MPC) timing with 10x AT factor.

VI. Conclusion

Historically, privacy was almost implicit, because it was hard to find and gather information. But in the digital world, whether it's digital cameras or satellites or just what you click on, we need to have more explicit rules - not just for governments but for private companies. - Bill Gates [84]

6.1 Summary

How private data is used is changing rapidly. It is becoming easier for private companies and governments to get the data and researchers are developing new ways to use the data. Some of the benefits to increased usage of private data are explored in this dissertation. Some privacy risks are well understood. Many, however, are not immediately obvious.

Consider a recent example of how privacy risks were not understood until after the fact. In 2006, Netflix announced a contest with a one million dollar prize. The goal of the contest was to develop algorithms to recommend movies to Netflix users. To assist with the contest, Netflix released a dataset containing over 100 million movie ratings from over 400 thousand of its users. Netflix claimed that ”all customer identifying information has been removed” from the dataset. In their seminal work, Narayanan and Shmatikov showed this to not be the case, however. They found that by using publicly available information on the internet, they were able to de-anonymize much of the Netflix dataset.

The Netflix example is only one in a series of such de-anonymization attacks. Other examples include de-anonymizing Massachusetts hospital discharge data [85], de-anonymizing DNA sequence datasets [86], and many others.

While many solutions exist to such problems, the technological advances discussed in this dissertation focus on privacy-preserving computation techniques. At a high level, privacy-preserving computation means that inputs to a computation are kept private from all except the original owner of the data and the only additional information that is learned by any party is the output of the computation (and the information that can be directly inferred by that). A major benefit of privacy-preserving computation techniques is that the privacy guarantees are highly formalized. Compare this with anonymization techniques used by Netflix and in other scenarios, where the privacy guarantees are not formalized.

6.2 Contributions

This dissertation presents a new paradigm for privacy-preserving computation, transferable multiparty computation (T-MPC). In this paradigm, the parties running the computation are allowed to change over time, while still maintaining high security and privacy requirements. Chapter 3 presents protocols for T-MPC in both the honest-but-curious (HbC) and the malicious adversary models. These protocols result in much more efficient and scalable privacy-preserving smart metering. Under a smart metering application, T-MPC enables network sizes that are orders of magnitude larger that was previously possible. This helps solve a significant barrier in deploying smart metering.

Under another application, decentralized reputation systems, T-MPC is used to significantly increase information availability. Information availability is crucial in decentralized reputation systems, as without reputation information, the system is useless. T-MPC enables privacy-preserving delegation of reputation information in such systems, something that had never before been achieved.

When deploying a privacy-preserving system in the real-world, system designers must make decisions relating to the security of the protocols they choose to deploy. Often this is defined by the adversary model that a given protocol is proven secure under. The analysis in this dissertation shows that T-MPC can greatly reduce the overhead of using more secure malicious model protocols. Malicious model protocols have often been believed to be far too inefficient for real-world use. T-MPC has changed this.

6.3 Recommendations for Future Research

Chapter 1 discusses a number of high-level areas in which researchers are applying privacy-preserving computation techniques to solve important privacy-related issues. This illustrates the broad nature of privacy concerns and the state of practical privacy-preserving computation. Future research conducted in the areas of theory and application are necessary to further adoption and use in industry. The following are of particular interest, however:

1. **Protocols:** Recent advances in MPC have focused on the case of a dishonest majority. The T-MPC protocols in this dissertation focus on honest majority scenarios, with the exception of the protocol specified in Section 4.2, which falls under the HbC model. A significant advancement would be to apply the T-MPC paradigm to a recent dishonest majority protocol such as SPDZ [43]. The foundation for such a T-MPC enhancement could come from the work in Section 4.2 as both use additive secret sharing, but the adaptation is non-trivial.

2. **Applications:** Any of the application domains outlined in Chapter 1 would be of great interest for applying T-MPC, and there could be significant gains in any of these using T-MPC. One additional area of interest is outsourced privacy-preserving computation. As more computation moves to the cloud, privacy issues begin to become very real. T-MPC could be used to enhance availability of such a service by privately transferring information as computation servers go offline. Another interesting application of T-MPC related to this application domain is in computation hopping. In other words, the set of servers running the computation changes over time so an attacker has a hard time knowing who to attack.

3. **Programming Constructs:** Much of the privacy-preserving computation literature assumes that computations are specified as (boolean or arithmetic) circuits. This requires a lot of domain-specific knowledge by the implementer. Researchers have

developed methods to alleviate this by providing constructs that better reflect typical development environments. As there will likely be specific construct relating to T-MPC (e.g., when to transfer, when is the optimal point in a program to transfer, etc), a number of significant contributions could be made to allow developers to build applications that use T-MPC using programming constructs they are already familiar with.

Bibliography

[1] M. A. Lisovich, D. K. Mulligan, and S. B. Wicker, "Inferring personal information from demand-response systems," *IEEE Security and Privacy*, vol. 8, pp. 11–20, Jan. 2010.

[2] A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin, "Private memoirs of a smart meter," in *2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys 2010)*, (Zurich, Switzerland), November 2010.

[3] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, (Berlin, Heidelberg), pp. 223–238, Springer-Verlag, 1999.

[4] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology*, pp. 10–18, Springer, 1985.

[5] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[6] C. Gentry, *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.

[7] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the aes circuit," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, 2012.

[8] A. C. Yao, "Protocols for secure computations," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pp. 160–164, IEEE, 1982.

[9] A. Yao, "How to generate and exchange secrets (extended abstract)," *Foundations of Computer Science*, pp. 162–167, 1986.

[10] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the Ninteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pp. 218–229, ACM, 1987.

[11] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pp. 1–10, ACM, 1988.

[12] I. Damgard, M. Keller, E. Larraia, C. Miles, and N. Smart, "Implementing aes via an actively/covertly secure dishonest-majority mpc protocol." Cryptology ePrint Archive, Report 2012/262, 2012. http://eprint.iacr.org/.

[13] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges." Cryptology ePrint Archive, Report 2010/543, 2010. http://eprint.iacr.org/.

[14] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology–EUROCRYPT 2005*, pp. 457–473, Springer, 2005.

[15] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "Reusable garbled circuits and succinct functional encryption." Cryptology ePrint Archive, Report 2012/733, 2012. http://eprint.iacr.org/.

[16] A. Sahai and B. Waters, "Attribute-based encryption for circuits from multilinear maps." Cryptology ePrint Archive, Report 2012/592, 2012. http://eprint.iacr.org/.

[17] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, , and N. Zeldovich, "How to run turing machines on encrypted data." Cryptology ePrint Archive, Report 2013/229, 2013. http://eprint.iacr.org/.

[18] M. Naveed, S. Agrawal, M. Prabhakaran, X. Wang, E. Ayday, J.-P. Hubaux, and C. Gunter, "Controlled functional encryption," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, (New York, NY, USA), pp. 1280–1291, ACM, 2014.

[19] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung, "Multi-authority secret-ballot elections with linear work," in *Advances in CryptologyEUROCRYPT96*, pp. 72–83, Springer, 1996.

[20] J. Benaloh, *Verifiable secret-ballot elections*. PhD thesis, Yale University, 1987.

[21] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, "Practical multi-candidate election system," in *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pp. 274–283, ACM, 2001.

[22] B. Adida, "Helios: Web-based open-audit voting.," in *USENIX Security Symposium*, vol. 17, pp. 335–348, 2008.

[23] J. Schiller and A. Voisard, *Location-based services*. Elsevier, 2004.

[24] G. Yavaş, D. Katsaros, Ö. Ulusoy, and Y. Manolopoulos, "A data mining approach for location prediction in mobile environments," *Data & Knowledge Engineering*, vol. 54, no. 2, pp. 121–146, 2005.

[25] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung, "Mining, indexing, and querying historical spatiotemporal data," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 236–245, ACM, 2004.

[26] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *Proceedings of the 18th international conference on World wide web*, pp. 791–800, ACM, 2009.

[27] C. Bettini, X. S. Wang, and S. Jajodia, "Protecting privacy against location-based personal identification," in *Proceedings of Secure Data Management*, pp. 185–199, Springer, 2005.

[28] G. Zhong, I. Goldberg, and U. Hengartner, "Louis, lester and pierre: Three protocols for location privacy," in *Privacy Enhancing Technologies*, pp. 62–76, Springer, 2007.

[29] A. Solanas and A. Martínez-Ballesté, "Privacy protection in location-based services through a public-key privacy homomorphism," in *Public Key Infrastructure*, pp. 362–368, Springer, 2007.

[30] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location privacy via private proximity testing.," in *NDSS*, 2011.

[31] R. Gelles, R. Ostrovsky, and K. Winoto, "Multiparty proximity testing with dishonest majority from equality testing." Cryptology ePrint Archive, Report 2012/378, 2012. http://eprint.iacr.org/.

[32] J. Vanacek, "How will big data remake medicine?," April 2014. http://www.forbes.com/sites/sap/2014/04/03/how-will-big-data-remake-medicine/.

[33] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik, "Privacy preserving error resilient dna searching through oblivious automata," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 519–528, ACM, 2007.

[34] M. Blanton and M. Aliasgari, "Secure outsourcing of dna searching via finite automata," in *Proceedings of Data and Applications Security and Privacy XXIV*, pp. 49–64, Springer, 2010.

[35] F. Bruekers, S. Katzenbeisser, K. Kursawe, and P. Tuyls, "Privacy-preserving matching of dna profiles.," *IACR Cryptology ePrint Archive*, vol. 2008, p. 203, 2008.

[36] Y. Chen, B. Peng, X. Wang, and H. Tang, "Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds.," in *NDSS*, 2012.

[37] M. Barni, P. Failla, R. Lazzeretti, A. Paus, A. Sadeghi, T. Schneider, and V. Kolesnikov, "Efficient privacy-preserving classification of ecg signals," in *Information Forensics and Security, 2009. WIFS 2009. First IEEE International Workshop on*, pp. 91–95, IEEE, 2009.

[38] M. Barni, P. Failla, R. Lazzeretti, A. Sadeghi, and T. Schneider, “Privacy-preserving ecg classification with branching programs and neural networks,” *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 2, pp. 452–468, 2011.

[39] M. R. Clark and K. M. Hopkinson, “Towards an understanding of the tradeoffs in adversary models of smart grid privacy protocols,” in *Power and Energy Society General Meeting, IEEE*, pp. 1–5, 2013.

[40] O. Goldreich, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[41] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “Fully homomorphic encryption without bootstrapping.” Cryptology ePrint Archive, Report 2011/277, 2011.

[42] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[43] I. Damgard, V. Pastro, N. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption.” Cryptology ePrint Archive, Report 2011/535, 2011. http://eprint.iacr.org/.

[44] Y. Desmedt and S. Jajodia, “Redistributing secret shares to new access structures and its applications,” Tech. Rep. ISSE TR-97-01, George Mason University, Fairfax, VA, July 1997.

[45] T. M. Wong, C. Wang, and J. M. Wing, “Verifiable secret redistribution for threshold sharing schemes,” tech. rep., Carnegie Mellon University, Pittsburgh, PA, 2002.

[46] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols.” Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org.

[47] M. Geisler, *Cryptographic Protocols: Theory and Implementation*. PhD thesis, Aarhus University, 2010.

[48] R. Cramer, I. Damgård, and J. B. Nielsen, “Secure multiparty computation and secret sharing – an information theoretic approach.” http://www.daimi.au.dk/~ivan/MPCbook.pdf.

[49] D. E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[50] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen, “Asynchronous multiparty computation: Theory and implementation,” in *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC ’09*, pp. 160–179, Springer-Verlag, 2009.

[51] R. McEliece and D. Sarwate, “On sharing secrets and Reed-Solomon codes,” *Communications of the ACM*, vol. 24, no. 9, pp. 583–584, 1981.

[52] B. Krebs, "FBI: Smart Meter Hacks Likely to Spread." Krebs on Security, April 2012. http://krebsonsecurity.com/2012/04/fbi-smart-meter-hacks-likely-to-spread/.

[53] A. Connelly, "Using SDR to Read Your Smart Meter." Hack a Day, February 2014. http://hackaday.com/2014/02/25/using-sdr-to-read-your-smart-meter/.

[54] Z. Erkin, J. R. Troncoso-Pastoriza, R. L. Lagendijk, and F. Perez-Gonzalez, "Privacy-preserving data aggregation in smart metering systems," *IEEE Signal Processing Magazine*, vol. 30, pp. 75–86, 2013.

[55] Z. Erkin and G. Tsudik, "Private Computation of Spatial and Temporal Power Consumption with Smart Meters," in *Applied Cryptography and Network Security - 10th International Conference*, 2012.

[56] F. D. Garcia and B. Jacobs, "Privacy-friendly Energy-metering via Homomorphic Encryption," in *Proceedings of the 6th International Conference on Security and Trust Management*, 2011.

[57] G. Danezis, M. Kohlweiss, and A. Rial, "Differentially Private Billing with Rebates," in *Proceedings of the 13th International Conference on Information Hiding*, 2011.

[58] E. Shi, P. U. C. Berkeley, T.-h. H. Chan, and D. Song, "Privacy-Preserving Aggregation of Time-Series Data," in *Proceedings of the Network and Distributed System Security Symposium*, 2011.

[59] F. Li, B. Luo, and P. Liu, "Secure Information Aggregation for Smart Grids Using Homomorphic Encryption," in *First IEEE International Conference on Smart Grid Communications*, pp. 327–332, 2010.

[60] K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," in *Proceedings of the tenth international conference on Information and knowledge management*, pp. 310–317, ACM, 2001.

[61] A. Jsang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th bled electronic commerce conference*, pp. 41–55, 2002.

[62] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of the 12th international conference on World Wide Web*, pp. 640–651, ACM, 2003.

[63] S. Buchegger and J.-Y. Le Boudec, "A robust reputation system for peer-to-peer and mobile ad-hoc networks," in *P2PEcon 2004*, no. LCA-CONF-2004-009, 2004.

[64] E. Pavlov, J. S. Rosenschein, and Z. Topol, "Supporting privacy in decentralized additive reputation systems," in *Trust Management*, pp. 108–119, Springer, 2004.

[65] O. Hasan, L. Brunie, and E. Bertino, "Preserving privacy of feedback providers in decentralized reputation systems," *Computers & Security*, vol. 31, no. 7, pp. 816–826, 2012.

[66] T. Dimitriou and A. Michalas, "Multi-party trust computation in decentralized environments," in *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pp. 1–5, IEEE, 2012.

[67] S. Dolev, N. Gilboa, and M. Kopeetsky, "Computing multi-party trust privately: in o (n) time units sending one (possibly large) message at a time," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 1460–1465, ACM, 2010.

[68] O. Hasan, L. Brunie, E. Bertino, and N. Shang, "A decentralized privacy preserving reputation protocol for the malicious adversarial model," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 6, pp. 949–962, 2013.

[69] J. Launchbury, I. S. Diatchki, T. DuBuisson, and A. Adams-Moran, "Efficient lookup-table protocol in secure multiparty computation," in *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, (New York, NY, USA), pp. 189–200, ACM, 2012.

[70] I. de Jong, "Pyro - python remote objects - 4.26," 2014. https://pythonhosted.org/Pyro4/.

[71] G. Ács and C. Castelluccia, "I have a dream!: Differentially private smart metering," in *Proceedings of the 13th International Conference on Information Hiding*, IH'11, Springer-Verlag, 2011.

[72] G. Danezis, M. Kohlweiss, and A. Rial, "Differentially private billing with rebates," in *Proceedings of the 13th international conference on Information Hiding*, IH '11, pp. 148–162, Springer, 2011.

[73] C. Thoma, T. Cui, and F. Franchetti, "Secure multiparty computation based privacy preserving smart metering system," in *North American power Symposium*, pp. 1–6, 2012.

[74] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys." Cryptology ePrint Archive, Report 2013/013, 2013. http://eprint.iacr.org.

[75] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified vss and fast-track multiparty computations with applications to threshold cryptography," in *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pp. 101–111, ACM, 1998.

[76] Z. Beerliová-Trubíniová and M. Hirt, "Perfectly-secure mpc with linear communication complexity," in *Proceedings of the 5th conference on Theory of cryptography*, TCC'08, pp. 213–230, Springer, 2008.

[77] R. Cramer, I. Damgård, and Y. Ishai, "Share conversion, pseudorandom secret-sharing and applications to secure computation," in *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, Spring-Verlag, 2005.

[78] S. McLaughlin, D. Podkuiko, and P. McDaniel, "Energy theft in the advanced metering infrastructure," in *Proceedings of the 4th International Conference on Critical Information Infrastructures Security*, 2009.

[79] S. McLaughlin, D. Podkuiko, A. Delozier, S. Miadzvezhanka, and P. McDaniel, "Embedded firmware diversity for smart electric meters," in *Proceedings of the 5th USENIX Workshop on Hot Topics in Security (HotSec 2010), Washington DC*, 2010.

[80] M. J. Atallah, E. D. Bryant, and M. R. Stytz, "A Survey of Anti-Tamper Technologies," *CrossTalk*, pp. 12–16, Nov. 2004.

[81] M. Arora, P. Bhargava, and S. Pickering, "Mcf51em256 anti tamper features - a leap towards robust smart metering solutions." http://www.freescale.com/files/industrial/doc/brochure/BRMETERING__INT_ISSUE_4_09.pdf, 2009.

[82] J. Cappaert, B. Preneel, B. Anckaert, M. Madou, and K. D. Bosschere, "Towards tamer resistant code encryption: Practice and experience," in *Proceedings of the 4th International Conference on Information Security Practice and Experience*, ISPEC'08, pp. 86–100, Springer-Verlag, 2008.

[83] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pp. 290–299, ACM, 2003.

[84] S. Levy, "Bill Gates and President Bill Clinton on the NSA, Safe Sex, and American Exceptionalism," November 2013. http://www.wired.com/2013/11/bill-gates-bill-clinton-wired/2/.

[85] L. Sweeney, "Weaving technology and policy together to maintain confidentiality," *The Journal of Law, Medicine & Ethics*, vol. 25, no. 2-3, pp. 98–110, 1997.

[86] B. Malin and L. Sweeney, "How (not) to protect genomic data privacy in a distributed network: using trail re-identification to evaluate and design anonymity protection systems," *Journal of biomedical informatics*, vol. 37, no. 3, pp. 179–192, 2004.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 26–03–2015	**2. REPORT TYPE** Doctoral Dissertation	**3. DATES COVERED** *(From — To)* Jun 2011–Mar 2015

4. TITLE AND SUBTITLE The Theory and Application of Privacy-preserving Computation	**5a. CONTRACT NUMBER** **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Clark, Michael R.	**5d. PROJECT NUMBER** **5e. TASK NUMBER** **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB, OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENG-DS-15-M-013
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank	**10. SPONSOR/MONITOR'S ACRONYM(S)** **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Privacy is a growing concern in the digital world as more information becomes digital every day. Often the implications of how this information could be exploited for nefarious purposes are not explored until after the fact. The public is becoming more concerned about this. This dissertation introduces a new paradigm for tackling the problem, namely, transferable multiparty computation (T-MPC). T-MPC builds upon existing multiparty computation work yet allows some additional flexibility in the set of participants. T-MPC is orders of magnitude more efficient for certain applications. This greatly increases the scalability of the sizes of networks supported for privacy-preserving computation.

15. SUBJECT TERMS

Cryptography, privacy, multiparty computation, smart grid, reputation, trust

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Kenneth M. Hopkinson(ENG)
U	U	U	UU	117	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-3636 x4579 kenneth.hopkinson@afit.edu