Secure Multi-Party Computation in Cloud Services*

An Analysis on Identifying Bottlenecks and Improving Performance, Computation, and Bandwidth†

Kevin K. Pho‡
California Academy of Mathematics and Science
Carson, CA, Los Angeles Country
kevinkhoapho@gmail.com

ABSTRACT

Cloud computing is on the rise. With more data being uploaded to remote servers, privacy becomes a growing concern. It is important to develop new protocols to ensure the security of stored data. One such method is secure multi-party computation (MPC). However, although it can be considered to be the most viable solution presently, it is still lacking in efficiency. Many developments have been made, but research must be made to identify the bottlenecks. Adapting Huang, Evans, Katz, and Malka's [14] methodology, an analysis was made within the time domain (speed), memory domain (bandwidth), and computational cost domain to identify where improvements need to be made to help progress MPC enough to be used practically in real-world servers. The framework simulating two parties, Alice and Bob, reveals that the garbled circuit should be optimized to reduce the amount of oblivious transfers despite its constant complexity. As such, targeting these bottlenecks in garbled circuits (GCs) for MPC could bring about new research to quicken the move to secure cloud networks [1] [2].

CCS CONCEPTS

• **Security and privacy → Cryptography**; *Formal security models*; *Privacy-preserving protocols*; Distributed systems security; Management and querying of encrypted data; Privacy protections; • **Networks → Cloud computing**;

KEYWORDS

AP Research, secure multi-party computation, cloud services, oblivious transfer, garbled circuits

*The research seeks to analyze MPC to safeguard privacy in cloud networks; the word count is around 5613 words

†The analysis is rather limited but extensible

‡The original author's name and any referenced individuals/locations were redacted for anonymity reasons for submission to College Board™

[1]This benefits both users and cloud service providers (CSPs)

[2]There is no publication; therefore, the ACM ISBN and DOI are blank

RESEARCH'18, April 2018, Carson, CA, Los Angeles Country

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Kevin K. Pho. 2018. Secure Multi-Party Computation in Cloud Services: An Analysis on Identifying Bottlenecks and Improving Performance, Computation, and Bandwidth. In *Proceedings of College Board™ AP Research (RESEARCH'18).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Literature Review

1.1.1 Background. Cloud computing is on the rise: a global phenomenon in the modern computing era, affecting a large subset of the human population. Cloud networking is growing rapidly in popularity amongst both consumers and enterprises as they are moving to cloud services over local services, wishing to take advantage of these servers when using services, such as data storage or file processing. Such benefits of cloud computing continue, ranging from cost, scalability, reliability, mobility, and elasticity (ready access to resource pools) [6] to computational power, speed, hardware, software, and storage space [13].

This rise in demand is growing as seen by the rise in supply. Cloud service providers (CSPs) go from Google™, Salesforce™, YouTube™, Facebook™, and Myspace™ [26]. And, these companies are prospering. For example, Alphabet™ [1], the parent company of Google™, has reported (through Ruth Porat, the CFO) revenues of $26 billion and a growth of 21% between the second quarters of 2017 versus 2016. Within a year, the company has grown largely, which has a strong SaaS, PaaS, and IaaS, all being cloud platforms (defined below). Google™ is known for their software in cloud technologies, such as Google Drive™ to Google Cloud™'s offerings to host services for others. In fact, their large cloud networks have enough processing power to provide real-time editing and collaboration for millions of users. They even specialize in analytics of Big Data (defined below), such as for advertising and marketing. Evidently, this large short-term growth is a symptom of the long-term trend of cloud technology, as Google™ has accumulated millions of users and trillions of gigabytes of data through their cloud, siince its release.

CSPs provide access to these cloud networks in many ways. Many researchers [10, 23, 26] have characterized them as the following abbreviations:

B2B = Business to Business when two businesses work together, such as two cloud networks sharing data

B2C = Business to Customer when a client interacts, often providing data, with a business's service

SaaS = Software as a Service a single application in the cloud, often performing operations on data, such as Google Drive™ or Dropbox™

PaaS = Platform as a Service a single platform in the cloud, providing users the ability to, say, store a database or have partial control of an OS to run a program under a runtime

IaaS = Infrastructure as a Service a single server in the cloud, allowing users to own and control their own server within constraint, such as Amazon Web Services™

These reflect the many models of cloud services; in particular, SaaS is the most user-friendly service, which includes Google Docs™ and Google Drive™. Their easy accessibility has resulted in many customers using their services, which perform computations on the cloud for a user.

To explain the aforementioned phenomenon, Big Data is the trend where cloud services are accumulating data at an exponential rate, in which the services now have the capacity to serve large amounts of data by personalizing their services or even process the data for analytics to best understand their customers (e.g. to identify market trends, train neural networks, et cetera) [13] As Hashem et al. [13] state, "[c]loud computing and [B]ig [D]ata are conjoined."

Overall, the growth and evolution of the cloud is undeniable. As such, the growth of cloud computing—or, better, trusted remote computation—indicates a potential for large-scale expansion. In particular, fog computing via the "Internet of Things" has become a novel innovation within cloud computing where the "cloud" has become an omnipresent "fog." The Internet of Things is composed of different networked devices (i.e.peer-to-peer or decentralized networks, such as a piconet), including smart home devices to even smart lights. These devices can communicate to each other to best monitor and improve one's life.

1.1.2 Problem. However, the growth of cloud computing poses a unique, growing threat to privacy: more and more users are trusting third-party services with their data without any privacy guarantees. In other words, the data is secured by "trust." Ryan [22] explains that private data must be safeguarded by these third-parties as more people rely on the increasing amount of services, such as Google Docs™. He pinpoints that, in order to manipulate any data sent by a client to the server, that data must be decrypted first, leaving a window where the data is left vulnerable, which creates the potential for an infiltration of the servers and a massive data leak if the system were to be compromised. The data may be released in a leak, for possibly malicious purposes in data mining, Big Data computations, extortions, et cetera. Despite the offering of privacy by cloud service providers (CSPs) that should safeguard data from unwarranted, prying eyes, data theft still occurs. Often, in order to keep the data operations efficient, the data are often left unencrypted on these servers, only defended by the security of trust instead of mathematical/cryptographic principles which guarantee perfect security. Even worse, even the company in itself may access these records to sell or use without permission.

What is worse is that users often have no control of their data once they are on these remote servers [21]. Within a B2C network, a user within his/her local network remotely connects to the cloud to another network, being the CSP. There is direct communication between the two parties; however, there is no trusted third-party in the middle to mediate all exchanges. As such, nothing ensures that the data will remain same. In other words, control of the data is delegated to whomever has access to or owns the data. The same applies for a B2B network, where two businesses would want to share data to benefit off the resultant output, such as a combined intersection of their data to learn more about their shared customers, but the question remains: who maintains fairness and privacy?

The issues could have such long-lasting impacts. Sensitive data may be collectively stored on these servers: federal documentation, medical records, private messages, browsing history, et cetera. This data are at large risk because a malicious insider party can use the private data for a variety of reasons, including to extort users, steal data, sell data, illegally use data, defame a user. The vulnerable data can even be attacked from the outside and stolen in leaks. If left unsolved, privacy risks will grow as more and more data is purged into the cloud.

And, a solution is needed quickly. Besides the strong trend of growth in the cloud industry, cloud computing is likely to grow again under the aforementioned fog computing. As Bonomi et al. [3] explained in 2012, fog computing has a "strong presence of streaming and real-time applications" while also having the potential to be mobile (e.g. smartwatches). As these are often equipped with many sensors, malicious hackers may not only be able to receive static data but be able to stream live sensor data (i.e. be undesirably used in the background). This is a major threat to privacy as these sensors, including cameras and microphones, can watch one's private life at home. With an ever increasing growth of cloud computers in the world and "Big Data" being on the rise as more corporations utilize cloud-like services to store large amounts of data on their consumers (to which they may want to perform general computations on the dataset to find trends in their market), it is important to mitigate the privacy risks before the growing cloud grows too large to deploy a solution or even allow recovery.

Additional implications of an unresolved privacy problem go on to even limit joint computations. If the data must be encrypted, current technologies would prevent CSPs from running joint computations over the private dataset for the benefit of the user, such as to analyze medical scans to diagnose the user or training autonomous vehicles using satellite data or driver data. Even the cloud itself is threatened. Kuyoro and his research team [17] state that "[s]ecurity is one of the major issues which hamper the growth of [the] cloud," explaining how users would be vigilant in handling data to some third-party company, where data breaches would be a potential risk. In addition, it carries along many legal issues regarding security and user privacy as the data are outsourced to third-parties [7, 25]. L. M. Kaufman explains how this virtual environment requires corporate policies and security methods to be made as litigation regarding the cloud is still in progress (e.g. who is responsible for the security of the data) [16]. Such legal cases include backlash on Facebook for not ensuring that data be secured while being used or deleted after being used by services on Facebook platforms, or the lawsuits on Snapchat for massive data leaks. To find some solution, one must carefully consider "confidentiality, integrity, availability, accountability, and privacy-preservability" are five attributes to consider with respect to attacks, threat models, and defense [25]. In other words, when designing protocols, one

must ensure that the data remain private and untampered. Overall, the literature has indicated the need to address the privacy problem.

1.1.3 Proposed Solutions. Solutions include a deletion policy, which Ryan [22] explains cannot be guaranteed. Also, some researchers [5] acknowledge that tamper-proof hardware does not scale well. Itani et al. [15] suggests another solution: privacy as a service. This would be a new architecture to securely store and process user data using new protocols and cryptographic processors, protecting sensitive data from unauthorized access; although, this would incur costs and would not scale well due to the exponential nature of certain cryptographic primitives with respect to processing power.

However, cryptography is a viable solution, which even bypasses the legal concern to some extent (as responsibility falls mostly under the implementation of the hardware and software). As Ren states, developments in cryptography must be made before a trustworthy public cloud can be made [21].

Such an alternative in the form of a cryptographic development was homomorphic encryption to directly compute on encrypted data; however, this technology is nascent and too slow for commercial use, including in tests in Amazon Web Services™ [9]. For example, Naehrig et al. [19] suggest fully homomorphic encryption (FHE) schemes for performing some arbitrary computation on encrypted data; they explain how proposed schemes have been made and improved. Using the learning-with-errors (LWE) problem (standard, ring, et cetera) has been one recent breakthrough. One such proposal has been one using LWE efficiently for private information retrieval [4]. However, the efficiency of FHE is rather limited, requiring much progress to become practical. For example, LWE requires an evaluation depth to reduce the error to increase accuracy. Some protocols even use lattices with the short vector problem (SVP), a (usually) NP-hard problem, but this uses multidimensional arrays, which may impact speed and compactness [4]. This one protocol, proposed by Brakerski and Vaikuntanathan is efficient, but FHE is, in general, not considered to be efficient to be feasibly implemented in the near future as the field is still undergoing intensive research.

1.2 Hypothesis and Proposal

Yet, cryptography has not been completely ruled out. A solution is needed to protect user privacy and even enable novel joint computations. So, how can two untrusted parties work together? The output must be restricted to the participating parties too. If FHE were improved, privacy-preserving computations (PPC) would be possible. However, another PPC technique would be secure multi-party computation (MPC), called the most viable solution today by some [5]. Secure multi-party computation is a family of algorithms that enables the use of PPC for multiple parties. As usual, these protocols often have multiple steps. In order to ensure that the protocol is built securely, it must be built from cryptographic primitives that have already been proven secure mathematically, such as asymmetric encryption or practically as in one-way hash functions, where it is infeasible to attempt to crack a hash digest. To improve upon naive protocols, these protocols need to be efficient to reduce the performance gap [18]. The protocols must also be fair, correct, and private; creating a threshold for the number of corrupted parties is also important in the n-party case.

To define another term, privacy-preserving computations can protect the privacy of the input data while revealing the output to both parties. These can be extended to private auctions or even combining large databases, such as medical records, to facilitate management. This would be similar to a famous proposed problem. Yao [27] proposed in 1982 the famous Yao's Millionaire Problem: there are two millionaires who wish to compare each other's wealth but do not wish to share these values. Yao first proposed the garbled circuit (GC) construction here to map out some protocol π, which could be some computation for secret voting (counting the amount of values that match some candidate while maintaining privacy). He formally introduces the construction as two-party computation (a subset of secure MPC), generalized in 1986 [28] while describing OT protocols for private "knowledge transfer."

By enabling PPCs, the additional benefit of joint computations would allow companies to compute for the "medical, financial, and the advertising domains" (non-exhaustive), by allowing datasets on, say, patients to be combined or analyzed as a whole to find correlations or trends through machine learning (e.g. artificial neural networks) [19]. Furthermore, Agrawal et al. [2] explain how database management systems (DBMS) are crucial for the transition to cloud infrastructures in order to handle Big Data. Evidently, these systems are the ones that process the data; these store the data, which may be left encrypted or decrypted, so the DBMS also deals with encoding the data to maintain privacy. As such, these could be the services that run PPC.

Overall, many sources agreed that secure multi-party computation is currently the most viable solution for the cloud right now. For example, Goldreich [**?**] explains how information retrieval in the cloud can be performed by privacy-preserving computations (PPCs), such as secure multi-party computation (MPC). However, MPC is not commercially used in major cloud networks because it still is too slow for practical use; although, it is faster than FHE. There are many steps within the protocol, often repeated many times, such as symmetric encryptions. Oblivious transfer, one of the steps, uses a costly RSA-style encryption. There also appears to be a large bandwidth generated (most likely the circuit and truth table) by these protocols. Their ability to run on large data sets still remains unclear. As such, it is important to pinpoint the bottlenecks within these protocols to help advance the direction of MPC research as to reach optimization. Overall, research must be made to find out how to improve MPC protocols to best push forward MPC into the real world for practical use in the privacy problem.

1.3 Research Progression

As the research progressed, much evolution occurred, moving from the former question, which particularly focuses on how privacy preserving computations can be made.

In order to compensate for the privacy problem in cloud security, how can two parties selectively share information or run arbitrary computations without jeopardizing the privacy of the inputs?

However, this question has been explored many times with many proposed protocols without a strong, unified focus. Consequently

the research at hand evolved into finding the bottlenecks within an MPC protocol that affect performance, computation, and bandwidth. By finding ways to make MPC faster, lighter, and less costly, MPC is more likely to be used within the cloud, due to an analysis of the strengths and weaknesses of the raw primitives themselves.

What are the bottlenecks in performance, computation, and bandwidth within secure multi-party computation that are slowing down the application of MPC in the cloud?

2 METHOD

2.1 Philosophy

A quantitative analysis was chosen to be the best method for identifying bottlenecks in different protocols. This would give numbers to best compare the primitives within the protocols, and the numbers would reveal numeric correlations in the protocols to the input sizes, which helps show how a protocol begins to become less efficient as larger inputs are used. A qualitative interpretation of the data from a higher level would identify what steps are potential bottlenecks.

2.2 Framework

2.2.1 Interacting Parties. To understand the security of the protocol with respect to the interacting party, the research will consider the two-party case, assuming two parties named Alice and Bob. Alice is often considered to be some sort of server, with Bob interacting with her. There can be assumed to be a third-party, named Eve, who can attempt to eavesdrop in the middle of the communication to recover additional private information. As such, all secure data transfers must be encrypted (or, at least, within a threshold to prevent total recovery due to insufficient information collected; although, any data leakage is discouraged). To analyze some protocol π between the two parties, it is difficult to create some security proof; it is easier to use an equally secure simulation proof. It is important to distinguish between a security proof and simulation proof as is here. A security proof acts to show how some function is secure by using mathematical proofs. A simulation proof is an abstraction, and it requires one to show how a protocol would remain secure as in a simulation of a protocol.

An idea for this is that, to achieve a web of trust, the parties ought not to be trusted, but the web itself. Between any parties collaborating, they can consult some third-party to do the computation for them. However, that third-party will have control of their input data and even be able to deviate from the protocol, perhaps due to corruption by, say, bribery. What MPC intends to achieve is security by making the third-party itself a secure protocol to mediate communication exchanges.

2.2.2 Threat Models. To test these protocols, one should establish a threat model (simulation proof).

Ideal Model the protocol is a trusted, non-corruptible, and honest third-party

Semi-Honest Model (Passive Security) a security model in which some honest-but-curious attacker intends to gain further information out of the protocol but still cooperates and follows the protocol rules verbatim (no favor toward any one party); this can be done by collaboration

Malicious Model (Active Security) a security model in which some adversary / corrupted party intends to gain further information out of the protocol even by attacking the protocol internally or externally, often deviating from the protocol rules; this can be done by generation of a crafted input to crack the protocol

Of course, the ideal model only occurs in ideal simulations. The real versus ideal paradigm is important here. In the real world, one may encounter semi-honest adversaries or malicious adversaries, who will try to cheat the protocol in some direct or indirect manner. Malicious security is the best and also hardest security to achieve. However, for this research, we will only consider semi-honest security. This is due to the nature of certain MPC protocols, which require additional factors for improved security.

2.3 Design

2.3.1 Multi-Party Computation Design. A common way to run the PPC through MPC is by garbled circuits, as proposed by [27]. GCs work, from a high-level description, by having Alice generate a circuit and Bob evaluate said circuit. Some agreed-upon function f is modeled as a Boolean circuit, where it is composed of Boolean gates, which often take two inputs and results in an output based on a truth table mapping its behavior (often a comparison). This includes an AND gate to check if both bits are HIGH, resulting in a HIGH output and LOW output otherwise.

Each possible input combination to this truth table for f is encrypted with a unique symmetric key (AES encryption). The truth table of keys must be "garbled" to prevent arbitrary guessing. The keys are chosen based on Bob's inputs and recovered by Bob by oblivious transfer. Both inputs are combined (Alice sends her "random" keys as her input, revealing no information due to the scrambling), encrypted to get a new key to decrypt the output or continue the computation. Bob evaluates his own inputs and receives the output and decrypts it. Both parties receive their encrypted outputs (assuming Bob shares the output fairly) without knowing the other's inputs.

Garbled circuits have received some testing and developments. For example, such developments include constant-round garbled circuits (multi-party and fast over LAN as tested on computation of AES encryption) [24] and trapdoor-assumption-based garbled circuits (safe on a honest majority) [12]. For tests, researchers [8] have ran different implementations of AES using MPC, running different key lengths and protocols to analyze running time versus block size and count.

Oblivious transfer (OT), the aforementioned primitive, is a major stage in GCs. OT takes many forms. 1-2 OT is performed by Alice generating two masks for the choices, blinded by Bob; Alice decrypts both, to which Bob can only decrypt his desired choice. This principle is seen when Even, Goldreich, and Lempel [11] developed a 1-2 oblivious transfer (OT) protocol by running a modified RSA encryption-decryption scheme ("Rivest-Shamir-Adleman" is an asymmetric, public-key cryptography scheme, with separate encryption and decryption keys); both choices are essentially encrypted by Alice and masked by an encrypted "decision" by Bob

for Alice to later decrypt both, giving Bob the decryption of both, being able to read only the one he masked, essentially having both doubly lock and unlock the messages together.

The OT protocol to be used in this analysis is the one designed by Orlandi [20], being a simple protocol: a modified Diffie-Hellman key-exchange protocol (which, in summary, allows both parties to encrypt a common value and share a private key), where Alice sends Bob an encrypted value, which he encrypts with his own key and can choose to multiply a coefficient based on his choice), to which Alice decrypts both with and without this coefficient, such that Bob can decrypt later (the computational intractability of factoring these numbers keeps the values hidden, and the property of exponents allows these operations). The chosen OT algorithm was the latter due to simplicity of implementation. It should be noted that two-party OT can be generalized into n-party OT (repeated for different pairings of parties). Overall, oblivious transfer acts as the frequent inner stage with garbled circuits being the outer stage.

2.3.2 Expectations. Because many garbled circuits generate a boolean circuit, having to deal with each bit separately, a truth table generated would be exponentially large. The circuit should be minimized as it should be the reason per se for the performance, bandwidth, and computation cost; though, some researchers have targeted different parts of MPC protocols: secret sharing (for n-party MPC), oblivious transfer, et cetera.

2.3.3 Experimental Design. To verify/disprove these expectations, the design of the experiment was based on Huang, Evans, Katz, and Malka's methodology for a quantitative analysis [14]. The researchers benchmarked their garbled circuits protocol for time under the semi-honest threat model (assuming a adversary is not honest or is malicious but will still follow the protocol, unlike the malicious threat model), comparing how efficient and capable a protocol is by testing how fast (time to successfully finish) it runs on datasets of variable sizes (how speed scales with larger inputs) and how compact a protocol is by measuring much bandwidth is consumed by the protocol not only on the protocol as a whole but also on individual parts of the protocol (e.g. transfer, preprocessing, offline calculations, cryptographic primitives, et cetera) in order to discover performance bottlenecks for future improvements/iterations. Overall, the time complexity of the protocol as well as memory complexity can be determined. It would be optimal to discover a method that does not exponentially take longer or use more memory in order to privately run the computation or transfer the data, with larger datasets, as expected in real cloud servers. They used two-party computation using garbled circuits (which they agree is the most viable solution to the privacy problem) to evaluate string distance, genome alignment, and AES using modules of Boolean gates (i.e. to map out encryption sub-operations within AES to, say, shuffle a bit matrix) to provide security against malicious adversaries, creating systematic trust through security and not reputation.

For the research and to identify bottlenecks, the time domain (of different stages of MPC), memory domain (bandwidth sent by public/private channels), and computational-cost domain (number of encryptions/decryptions/hashes by each party) will be analyzed by timing the computation, counting the size of the messages, and counting the number of encryptions/decryptions/and hashes. Ten (10) trials were ran for each bit size, which is the input. The input was randomly generated using Python's pycrypto library and then fed into the OT protocol or to a GC for an n-bit comparison function, which consists of an AND gate for each input bit.

Table 1: Execution Time versus Bit Size for Oblivious Transfer

Bit Size (n) $[bit]$	Alice Time (t_a) $[s]$	Bob Time (t_b) $[s]$
8	4.935739279	4.963514566
16	5.019064188	5.067161083
32	5.347528696	5.520970583
64	4.975233555	5.069985867

Table 2: Execution Time versus Bit Size for Garbled Circuits

Bit Size (n) $[bit]$	Alice Time (t_a) $[s]$	Bob Time (t_b) $[s]$
0	1.41e-05	0.009559154511
1	9.78E-06	0.0002007484436
2	0.1643898487	0.0003819465637
3	0.9528546333	0.000720500946
4	2.384231567	0.004591464996
5	2.231013298	0.1123363972
6	N/A	0.1339011192
7	N/A	2.4115839

An artificial Alice and Bob class, inheriting a Party class with static variables as class attributes, was created, each running asynchronously in different threads than the main thread, used to manage the execution. Both were able to communicate to each other by modifying the other's queue, which monitored how much data was transferred. Each was also able to asynchronously alert the other whenever the data was successfully transferred to move onto the next step of the computation using a "ready" flag, which the other would check after finishing its role at the current stage of the protocol so that both can move on at the same time synchronously.

The custom, multi-threaded framework consists of the two parties:

- Alice as the sender and circuit generator
- Bob as the receiver and circuit evaluator

The code was written in Python, using the Jupyter tool to assist with debugging. The framework has been made to be akin to two remote servers speaking to each other.

3 RESULTS

3.1 Findings and Analysis

3.1.1 Time Domain. Oblivious transfer runs in O(1) or O(n) time with respect to bits; the results [3] from the code showed that oblivious transfer ran in constant or linear time (complexity). This is logical because oblivious transfer runs within a constant amount of rounds, which is primarily the data transfer. The size of the bandwidth data is the only significant variable that changes.

[3] The findings were averaged between the ten trials

```
def ObliviousTransfer1_2(self, target, m0, m1, bits):
    """Set up"""
    g = Util.number.getRandomInteger(bits)
    n = Util.number.getRandomInteger(bits)
    a = Util.number.getRandomInteger(bits)
    aMasked = 0

    while n == 0:
        n = Util.number.getRandomInteger(bits)
    while aMasked == 0:
        aMasked = pow(g, a, n)
        self.Encrypt(None, None, None)

    """Protocol"""
    self.SendWait(target, g, Party.BasePublic)
    self.SendWait(target, n, Party.BasePublic)
    self.SendWait(target, aMasked, Party.BasePublic)

    self.ReceiveWait(target, Party.BasePublic)
    bMasked = long(self.Buffer)
    k0 = self.Encrypt(Party.BaseHash, None, Util.number.long_to_bytes(pow(bMasked, a, n)))
    k1 = self.Encrypt(Party.BaseHash, None, Util.number.long_to_bytes(pow(bMasked / aMasked, a, n)))
    ke0 = self.Key(Party.BaseSymmetric, k0)
    ke1 = self.Key(Party.BaseSymmetric, k1)
    e0 = self.Encrypt(Party.BaseSymmetric, ke0, m0)
    e1 = self.Encrypt(Party.BaseSymmetric, ke1, m1)

    self.SendWait(target, e0, Party.BasePublic)
    self.SendWait(target, e1, Party.BasePublic)

    self.Return = True
```

Figure 1: High-Level Source Code of the OT Protocol for Alice

Garbled circuits (without the OT stage) runs in $O(2^n)$ time with respect to bits as seen by the data [3]. Alice spends more time (exponential complexity) to generate gates and keys, which grow with the bit size exponentially as she must create a table for every possible input. Bob does not run the OT but does run the decryptions (which are in $O(2^n)$ for every possible input). Bob, despite not having to run the exponential amount of oblivious transfers, still has to decrypt each gate with the inputs. The amount increases exponentially as the more amount of bits implies more decryptions and higher depth of the circuit. Though, many encryptions could be skipped; though, Bob will still have to scour the truth table to find his matching input.

Figure 3 graphs the data to demonstrate the exponential behavior of both parties for garbled circuits with somewhat strong confidence.

3.1.2 Memory and Computational Cost Domain. Computation and bandwidth increased with bit size by the end of the test [3] from around 4 messages (public channels) sent and around 8 encryptions (symmetric) (from 0 to 8 bits) to 250 (16632 bytes) sent and 50 (1024 bytes) received after encoding and to 100 encryptions and 100 hashes for Alice and 100 decryptions for Bob. Computational costs remain low as well as bandwidth for small inputs; although, these are exponential in nature too. For small cloud applications, garbled circuits remain viable. For larger applications, optimization of the circuit to reduce the impact of the exponential complexity are desired. Oblivious transfer remains constant; its frequency should be controlled (i.e. the circuit is the bottleneck) to reduce all three variables.

Overall, when increasing from 0 to 8 bits, having less than 10 encryptions jump to 100 encryptions as Bob has to evaluate a larger set of possible inputs; likewise, more data must be transferred as there is a larger circuit and, thus, table.

4 CONCLUSION

In conclusion, computational costs and bandwidth remain low for small inputs but increase exponentially. Therefore, MPC is viable for small cloud applications, but, to upscale to larger applications, the circuit must be optimized as the circuit itself was the bottleneck. oblivious transfer ran at a nearly constant time, so the best optimization is to reduce the amount of oblivious transfers needed to be ran. Overall, MPC can work well for small applications, but it does not scale well for larger applications, requiring much more optimization or developments due to the exponential growth of the

```
def ObliviousTransfer1_2(self, target, choice, bits):
    """Setup"""
    k = Util.number.getRandomInteger(bits)
    b = Util.number.getRandomInteger(bits)

    """Protocol"""
    self.ReceiveWait(target, Party.BasePublic)
    g = long(self.Buffer)

    self.ReceiveWait(target, Party.BasePublic)
    n = long(self.Buffer)

    self.ReceiveWait(target, Party.BasePublic)
    aMasked = long(self.Buffer)

    if (choice == 0):
        bMasked = pow(g, b, n)
        self.Encrypt(None, None, None)
    elif (choice == 1):
        bMasked = (aMasked * pow(g, b, n)) % n
        self.Encrypt(None, None, None)
    else:
        bMasked = 0

    self.SendWait(target, bMasked, Party.BasePublic)

    self.ReceiveWait(target, Party.BasePublic)
    e0 = str(self.Buffer)

    self.ReceiveWait(target, Party.BasePublic)
    e1 = str(self.Buffer)

    k = self.Encrypt(Party.BaseHash, None, Util.number.long_to_bytes(pow(aMasked, b, n)))
    ke = self.Key(Party.BaseSymmetric, k)
    if (choice == 0):
        output = self.Decrypt(Party.BaseSymmetric, ke, e0)
    elif (choice == 1):
        output = self.Decrypt(Party.BaseSymmetric, ke, e1)

    self.Return = True

    return output
```

Figure 2: High-Level Source Code of the OT Protocol for Bob

garbled circuit. Oblivious transfer does not appear to have such issues.

4.1 Limitations

However, it should be noted that the research was limited. Most MPC protocols were variants of the garbled circuit. This was worsened because there was no available universal MPC library, which means that there could have been errors in the implementation of the code. Furthermore, as the implementation was done manually, I was unable to test additional protocols that could have produced better security, nor test algorithms to run the circuits on. In addition, even though the computational environment was controlled, it would be best to run the framework on real cloud architectures to simulate real-world applications and also network latency within the protocol. Additional things to do include:

- Simulate latency [9]
- Correct possible errors in implementation of MPC protocols or test algorithms
- Correct possible design flaws in the test framework
- Ensure randomness in input

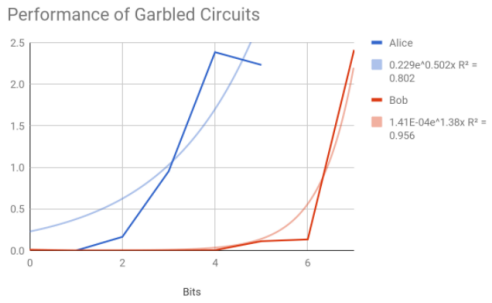

Figure 3: Time Complexity of the Two Parties

- Ensure accuracy of complexity measurements in the three domains
- Run more tests for increased precision (and allow longer running times)

4.2 Delimitations

The research was also limited in terms of scope. The delimitations include:

- Test alternatives to GCs
- Expand the scope of analysis to more algorithms as f
- GC optimization [4]
- Generalize to n-party computations using secret sharing
- Add security using commitment schemes
- Extend analysis into asymptotic behavior with higher bit sizes

4.3 Discussion and Implications

Overall, the privacy problem needs to be solved. Research into MPC and other PPCs is still novel. Much more time is needed for understanding of the technology to be strong enough to be used in practice. The benefits of improving MPC's efficiency, especially with smaller circuits would enable the average user to interact with CSPs securely and the CSP to interact with other CSPs for unique computations. MPC, if improved, will become lighter and easier to transport (bandwidth-wise), less expensive to run with fewer cryptographic operations, and faster to enable faster services to remove the overhead of PPC in secure cloud interactions, allowing users to benefit off cloud services, including in real-time applications. The cloud can continue to grow with minimal impact on user privacy, no longer needing to worry about the capitalization of data nor leaks of personal data, and allowing anonymous joint computations to enable better analytics.

4.4 Future of Research

As such, the future of this research is to continue investigating MPC protocols; in particular, to test optimization steps. It would also be important to extend this test in higher bit sizes to confirm the asymptotic behavior of the algorithm. There are additional questions to ask:

- Are there other feasible alternatives to garbled circuits?
- Is it worth devising a reusable circuit protocol at whatever computation cost it may incur?
- Could fully homomorphic encryption be more efficient?

As such, it is evident that secure multi-party computation requires further developments.

[4] Optimizations have also been made, such as Half-XOR, to cut down the size of the circuit

ACKNOWLEDGMENTS

The author would like to thank his/her AP Research teacher for guiding with the research and documentation process, helping organize the research into discrete tasks.

REFERENCES

[1] 2017. Alphabet Announces Second Quarter 2017 Results. Retrieved March 30, 2018 from https://abc.xyz/investor/news/earnings/2017/Q2_alphabet_earnings/

[2] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. 2011. Big Data and Cloud Computing: Current State and Future Opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT '11)*. ACM, New York, NY, USA, 530–533. https://doi.org/10.1145/1951365.1951432

[3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC '12)*. ACM, New York, NY, USA, 13–16. https://doi.org/10.1145/2342509.2342513

[4] Zvika Brakerski and Vinod Vaikuntanathan. 2014. Efficient Fully Homomorphic Encryption from (Standard) *LWE*. *SIAM J. Comput.* 43, 2 (2014), 831–871. https://doi.org/10.1137/120868669 arXiv:https://doi.org/10.1137/120868669

[5] Sven Bugiel, Stefan Nurnberger, Ahmad Sadeghi, and Thomas Schneider. 2011. Twin clouds: An architecture for secure cloud computing. In *Workshop on Cryptography and Security in Clouds (WCSC 2011)*, Vol. 1217889.

[6] M. Carroll, A. van der Merwe, and P. Kotzé. 2011. Secure cloud computing: Benefits, risks and controls. In *2011 Information Security for South Africa*. 1–9. https://doi.org/10.1109/ISSA.2011.6027519

[7] Daniele Catteddu. 2010. Cloud Computing: Benefits, Risks and Recommendations for Information Security. In *Web Application Security*, Carlos Serrão, Vicente Aguilera Díaz, and Fabio Cerullo (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 17–17.

[8] Ivan Damgård and Marcel Keller. 2010. Secure multiparty AES. In *International Conference on Financial Cryptography and Data Security*. Springer, 367–374.

[9] C. A. Dhote, M. M. Potey, and D. H. Sharma. 2016. Homomorphic encryption for security of cloud data. *Procedia Computer Science* 79 (2016), 175–181.

[10] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. [n. d.]. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing* 13, 18 ([n. d.]), 1587–1611. https://doi.org/10.1002/wcm.1203 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcm.1203

[11] Shimon Even, Oded Goldreich, and Abraham Lempel. 1985. A randomized protocol for signing contracts. *Commun. ACM* 28, 6 (1985), 637–647.

[12] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC '87)*. ACM, New York, NY, USA, 218–229. https://doi.org/10.1145/28395.28420

[13] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. 2015. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems* 47 (2015), 98 – 115. https://doi.org/10.1016/j.is.2014.07.006

[14] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster Secure Two-Party Computation Using Garbled Circuits.. In *USENIX Security Symposium*, Vol. 201. 331–335.

[15] W. Itani, A. Kayssi, and A. Chehab. 2009. Privacy as a Service: Privacy-Aware Data Storage and Processing in Cloud Computing Architectures. In *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*. 711–716. https://doi.org/10.1109/DASC.2009.139

[16] L. M. Kaufman. 2009. Data Security in the World of Cloud Computing. *IEEE Security Privacy* 7, 4 (July 2009), 61–64. https://doi.org/10.1109/MSP.2009.87

[17] S. O. Kuyoro, F. Ibikunle, and O. Awodele. 2011. Cloud computing security issues and challenges. *International Journal of Computer Networks* 3, 5 (2011), 247–55.

[18] Yehuda Lindell and Benny Pinkas. 2009. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality* 1, 1 (2009), 5.

[19] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can Homomorphic Encryption Be Practical?. In *Proceedings of the 3rd ACM Workshop on*

Cloud Computing Security Workshop (CCSW '11). ACM, New York, NY, USA, 113–124. https://doi.org/10.1145/2046660.2046682

[20] Claudio Orlandi. 2015. The Simplest Oblivious Transfer Protocol. (2015). https://simons.berkeley.edu/talks/claudio-orlandi-2015-06-10/ Securing Computation Talks/Symposium (Simons Institute for the Theory of Computing at the University of California, Berkeley; Summer 2015).

[21] Kui Ren, Cong Wang, and Qian Wang. 2012. Security challenges for the public cloud. *IEEE Internet Computing* 16, 1 (2012), 69–73.

[22] Mark D. Ryan. 2011. Cloud computing privacy concerns on our doorstep. *Commun. ACM* 54, 1 (2011), 36–38.

[23] H. Takabi, J. B. D. Joshi, and G. J. Ahn. 2010. Security and Privacy Challenges in Cloud Computing Environments. *IEEE Security Privacy* 8, 6 (Nov 2010), 24–31. https://doi.org/10.1109/MSP.2010.186

[24] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 39–56.

[25] Z. Xiao and Y. Xiao. 2013. Security and Privacy in Cloud Computing. *IEEE Communications Surveys Tutorials* 15, 2 (Second 2013), 843–859. https://doi.org/10.1109/SURV.2012.060912.00182

[26] Zhang Yandong and Zhang Yongsheng. 2012. Cloud computing and cloud security challenges. In *2012 International Symposium on Information Technologies in Medicine and Education*, Vol. 2. 1084–1088. https://doi.org/10.1109/ITiME.2012.6291488

[27] Andrew C Yao. 1982. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE, 160–164.

[28] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 162–167.