Optimal Transport Layer for Secure Computation

Markus Brandt[1,3], Claudio Orlandi[2], Kris Shrishak[1] and Haya Shulman[3]
[1]*Technical University of Darmstadt, Germany*
[2]*Department of Computer Science, DIGIT, Aarhus University, Denmark*
[3]*Fraunhofer Institute for Secure Information Technology SIT, Germany*

Keywords: Secure Two-party Computation, Garbled Circuits, Transport layer, Framework, Implementation.

Abstract: Although significantly improved, the performance of secure two-party computation (2PC) is still prohibitive for practical systems. Contrary to common belief that bandwidth is the remaining bottleneck for 2PC implementation, we show that the network is under-utilised due to the use of standard TCP sockets. Nevertheless, using other sockets is non-trivial: the developers of secure computation need to integrate them into the operating systems, which is a challenging task even for systems experts. To resolve the efficiency barrier of 2PC, we develop a framework, we call `Transputation`, which automates the integration of transport layer sockets into 2PC implementations. `Transputation` is the first tool which enables developers of 2PC protocols to easily identify and use the optimal transport layer protocol for the given computation task and network conditions. We integrate selected transport layer protocols into `Transputation` and evaluate the performance for a number of computational tasks. As a highlight, even a general purpose transport layer protocol, such as SABUL, improves the run-time of 2PC over TCP on EU-Australia connection for circuits with $> 10^6$ Boolean gates by a factor of 8. To enable evaluations of 2PC implementations in real life setups in the Internet we setup a distributed testbed.The testbed provides automated generation of network scenarios and runs evaluations of 2PC implementations. We evaluate `Transputation` on in different network setups and report on our experimental results in this work.

1 INTRODUCTION

Secure two-party computation (2PC) is a cryptographic tool that allows two remote parties to jointly compute any function on their inputs and obtain the result without leaking any other information. As more and more interaction is performed online between parties that do not trust (or only partially trust) each other, the need for secure and practically efficient 2PC solutions is increasing, and there are plenty of applications that secure 2PC could facilitate. The classic setting where 2PC is needed is that of two parties holding some data and gaining extra utility by combining their data with the data of the other party. However, the parties might not want (or might not be allowed) to share their data with each other due to privacy concerns, competition (e.g., financial) or legislation. In these settings 2PC is a powerful tool that enables collaborations that were previously infeasible. Examples of such applications include key management for digital currencies (Archer et al., 2018), auctions (Bogetoft et al., 2009), tax-fraud detection (Bogdanov et al., 2015), private set intersection (Pinkas et al., 2018) and even prevention of satellite collision (Hemenway et al., 2016).

Implementations Are Still Not Practical. Despite its huge potential, aside from initial attempts and scant success stories, 2PC still remains the focus of theoretical research. Most prototype implementations are meant to demonstrate feasibility (Halevi, 2018). The implementations are evaluated in simulated environments or on a single host without taking into account realistic network conditions nor the presence of other processes (Kreuter, 2017). Current 2PC implementations are not generally usable and the users have to tradeoff their privacy with efficiency by resorting to third parties for performing computations for them instead of running 2PC with the target service.

Computation Is Optimal. While the idea of 2PC is now more than 30 years old, the first public implementation of 2PC was released in 2004 (Malkhi et al., 2004). Since then, huge progress has been made in terms of protocol design and implementation engineering. The most efficient implementations of 2PC protocols are based on the protocol proposed in 1986 by Yao (Yao, 1986) which is built from garbled circuits (Bellare et al., 2012) and oblivious transfer (OT) (Naor and Pinkas, 2001).What has changed in recent years is that (thanks to protocol optimisations, hardware support for cryptographic operations

Brandt, M., Orlandi, C., Shrishak, K. and Shulman, H.
Optimal Transport Layer for Secure Computation.
DOI: 10.5220/0009826801300141
In *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications (ICETE 2020) - SECRYPT*, pages 130-141
ISBN: 978-989-758-446-6

and the use of multiple cores) the computation overhead of 2PC protocols has been reduced drastically and there are indications that the current constructions of 2PC based on garbled circuits have reached the theoretical lower bound (Zahur et al., 2015). It is now widely believed that the *bandwidth* and not the *computation*, is the remaining bottleneck in 2PC (Asharov et al., 2013; Zahur et al., 2015).

Communication Is yet to Be Explored. Although there has been work on reducing the communication complexity and limiting the amount of data that needs to be exchanged between the two parties, no research has been done on the networking layer of 2PC. Indeed, evaluations of 2PC implementations on a single machine now result in practical performance (Bellare et al., 2013; Schneider and Zohner, 2013). Nevertheless, when 2PC is evaluated on separate hosts, the latency overhead is prohibitive for practical applications (Nielsen et al., 2017; Wang et al., 2017). Evaluations of 2PC disregard issues faced by practical deployment of 2PC: many implementations are not evaluated in real life setups where diverse network conditions, such as packet loss, latency, and other traffic can impact performance (Kreuter, 2017; Halevi, 2018).

1.1 Contributions

In this work we explore two obstacles that have held back secure computation from being practical for real life systems. We identify two central issues which need to be resolved: (1) there should be an easy way for 2PC developers to integrate and experiment with various transport protocols in their implementations of 2PC and (2) there should be an automated way to evaluate and compare the performance of 2PC implementations in different network setups.

Automated Transport Layer. All 2PC implementations use the standard TCP socket supported in popular operating systems (OSes). We demonstrate through our evaluations that, in contrast to other transport protocols, TCP sockets result in high performance penalty and do not fully utilise the bandwidth. TCP connections not only suffer from poor performance on paths with high latency and packet loss but they also fail to adjust to rapidly changing network conditions and incur high latency with buffer bloat. Even in stable network conditions, TCP does not provide optimal performance, and is not suitable for different types of applications. In the recent years different variants of TCP and other transport protocols, tailored to different applications and network setups, have been proposed.

> *Why, then, are other more efficient transport layer protocols not used in implementations of secure computation?*

The reason is the difficulty of integrating new transport layer sockets. As a result, the developers use the default option of TCP. In order to use other transport layer sockets the developers have to integrate them into the OS kernel: this is a challenging task even for systems and networking experts. Hence, 2PC developers resort to using the default TCP sockets, supported in the available cryptographic implementations and operating systems.

> *Which transport layer protocol is optimal for 2PC implementations?*

To answer this question we perform evaluations with popular transport protocols. Our results demonstrate that there is no transport protocol that can provide optimal performance for all 2PC implementations. For optimal performance the transport protocol has to be selected as a function of the 2PC implementation, the size of the inputs, and the network conditions (e.g., packet loss, latency).

In this work we develop a transport layer framework for secure computation that we call `Transputation`, which automates the usage and integration of transport layer protocols into secure computation implementations. We use `Transputation` to demonstrate the performance improvements of other (even mainstream) protocols over TCP. For our evaluations we have integrated three transport protocols into `Transputation`: UDP, TCP and SABUL[1]. We explain the choice of the protocols in Section 4. During the computation, `Transputation` automatically identifies the most suitable transport protocol for a given computation task and network conditions. This adaptive behaviour on the transport layer consistently achieves high performance over different and complex real world network conditions.

Evaluations of 2PC. Due to the difficulty and the overhead of setting up real life testbed environments, 2PC developers typically evaluate the implementations on a single host or on simulated environments (Kreuter, 2017; Halevi, 2018). The performance in those setups is not representative of real networks, e.g., the developers cannot foresee the behaviour during execution with other traffic and in diverse network conditions. Although there are other platforms for running experiments in distributed setups, such as PlanetLab (Chun et al., 2003), they are

[1]Though we use the most updated version of SABUL, also commonly known as UDT or UDP-based Data Transfer Protocol, in this paper we use the former name to stress the difference with "regular" UDP.

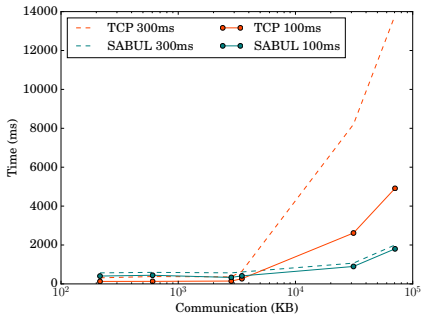

Figure 1: TCP vs. SABUL as garbled circuit size increases.

not tailored for 2PC evaluations and they do not support the various transport layer protocols, nor do they enable automated usage of transport protocols in 2PC implementations. Hence, the 2PC developers would themselves have to setup and install the requirements, including upload binaries, install libraries and dependencies, integrate transport layer protocols, measure latencies and packet loss.

For automating the evaluations of 2PC, we develop and set up a distributed 2PC testbed, which is integrated with `Transputation`, that provides a user friendly GUI and allows the users to select the network conditions as well as setup and run their 2PC implementations over the Internet with the required latency, packet loss and concurrent traffic. Our testbed is setup on `Vultr` cloud instances in different physical locations, over networks with varying latencies and packet loss and other concurrent traffic. By using our preinstalled implementations or by uploading binaries of their implementations, 2PC developers can perform real life evaluations and receive immediate results without the need to install or use any traffic monitoring tools. We provide the source code implementation of `Transputation` at https://github.com/Fraunhofer-SIT/transputation.

Through our evaluations using the platform, we demonstrate that even general purpose protocols already provide significant performance improvement over standard TCP sockets. For instance, for large circuit sizes in WAN setting, SABUL performs 8× better than TCP (Figure 1). Of course, transport protocols tailored for specific tasks would further improve efficiency. There is a large body research of showing performance improvements over general purpose transport protocols when tailoring protocols to specific tasks and engineering patches for specific network conditions, e.g., (Caini and Firrincieli, 2004; Ha et al., 2008; Obata et al., 2011; Wu et al., 2013; Alizadeh et al., 2011).

Organisation

In Section 2 we introduce the `Transputation` framework and its layers. The optimizations and implementations on secure computation layer and the transport layer are explained in Section 3 and 4 respectively. In Section 5 we present the design and implementation of the `Transputation` framework. In Section 6 we report our simulation and evaluation results of `Transputation` on our testbed environment. In Section 7 we review related work and we conclude this work in Section 8.

2 TRANSPUTATION FRAMEWORK

In this section we provide an overview of `Transputation`and explain how it can be used.

2.1 Components

The goal of `Transputation` is to identify which transport layer protocol is optimal for a given computational task at hand, the input sizes, the network setup and buffer sizes. The functionality of `Transputation` is split between two layers: the *secure computation* layer and the *transport* layer.

Secure Computation Layer, composed of secure computation implementations, accepts the function or application to be evaluated securely and the size of the circuit representing the function. In Section 3, we motivate our choices for 2PC implementations that we integrate into `Transputation`. We identify the parameters that impact the data volume and communication pattern, and deploy optimizations relevant for performance improvements on the transport layer.

Transport Layer is responsible to identify an optimal transport layer protocol for a given computation task. To that end, it determines the latency, bandwidth and packet loss on the network through live experimental probes. Transport layer is also responsible for avoiding overflow at the sender and the receiver. To avoid overflow, given the function to be computed and the size of inputs to the circuit, `Transputation` calculates the required buffer sizes on the sender and the receiver. We explain the transport layer implementation in Section 4.

We integrate, in both layers, popular and representative protocols to demonstrate the usage of framework as well as the evaluation of 2PC implementations. In the secure computation layer, we have integrated a garbled circuit protocol under standard assumptions (Gueron et al., 2015), a garbled circuit

protocol under circularity assumption (Zahur et al., 2015) and the scheme that assumes AES as a ideal cipher (Bellare et al., 2013). In the transport layer, we support TCP (Ha et al., 2008), UDP and SABUL (Gu and Grossman, 2008).

`Transputation` brings together the *secure computation* layer and the *transport* layer by choosing a suitable transport protocol based on the combination of parameters obtained from them. `Transputation` is built to support easy extensions at both layers. New implementations of secure computation as well as new transport layer protocols can be integrated into `Transputation`. The flexibility for easy integration of transport layer protocols is critical—new protocols are continually devised, e.g., to improve performance of applications in diverse network conditions or to support new Internet architectures. Future protocols may improve the performance of secure computation more than the existing options. `Transputation` provides agility and flexibility for 2PC developers enabling them to constantly extend the implementations with new transport protocols.

2.2 Usage of `Transputation`

`Transputation` makes it convenient for secure computation developers to focus on the 2PC protocol. First, the developers can implement the secure computation protocol without worrying about the underlying transport layer protocol. Then, the developers can invoke the *transport layer* part of `Transputation` which is implemented as a wrapper. Our implementation provides functions to set up a connection between the two parties, to send and to receive data.

We have implemented abstract classes into `Transputation` that makes selecting transport layer protocols simple and automated. Adding a new transport layer protocol to the framework only requires specifying a string to identify the protocol without requiring any changes to the secure computation implementation. Without the framework, substantial parts of the secure computation implementation will need to be re-written to accommodate new transport layer protocols. In many existing secure computation implementations, the code is not modular. The network code is interwoven with the application code such that not only is the transport layer protocol hard-coded into the application code, but also the IP-address and the port numbers are hard-coded. This has also been recently observed by (Halevi, 2018). Unwrapping such an implementation and adding transport layer protocols requires extensive rewriting of the code base. To begin with, an entangled code base prevents retrofitting of transport layer protocols. The code base will need to be made modular before considering the integration of different transport protocols.

Secure computation developers can use `Transputation` in three ways. First, they can use the *transport layer* module to automatically choose the most appropriate transport protocol in a given network setting for their secure computation implementation and 2PC application. The secure computation implementation is then integrated with the transport layer protocol which provides the most efficient run time. Second, the developer can test the secure computation protocol manually, by running the implementation using the various transport layer protocols available in `Transputation` under various network conditions to choose the most suitable transport layer protocol for the 2PC application. Finally, secure computation protocols consist of interactive primitives such as oblivious transfer (OT) and OT extensions. `Transputation` can be used not only by developers who use these primitives to construct a 2PC protocol, but also by the designers of the primitives to test the practicality of their design.

3 TWO-PARTY COMPUTATION LAYER

In this section we present the secure computation layer of `Transputation`. We explain our choice of 2PC protocols and the applications, describe the parameters that define the amount of data to be transmitted, the properties that impact the communication performance and we list recent optimisations relevant for our improvements on the transport layer, which we integrate into `Transputation`.

Currently most efficient technique for secure two-party computation is based on Yao's protocol with garbled circuits. The main features of Yao's protocol (as opposed to other protocols) are: it has constant rounds (in fact, it is round optimal when using a two-message oblivious transfer protocol), it is computationally cheap (since it mostly uses lightweight symmetric-key operations, and very few expensive public-key operations such as exponentiations). The main alternative to Yao's protocol is the GMW protocol (Goldreich et al., 1987) (which is purely based on Oblivious Transfer). Unlike Yao's protocol, the GMW protocol has non-constant round complexity (proportional to the depth of the circuit to be evaluated), and is less efficient than Yao. Hence, we do not consider it in this work.

We focus on the recently optimized garbling schemes. The three main characteristics of such garbling schemes are: the size of the produced garbled

circuits (which impacts the communication efficiency of the protocol), the number of encryptions/decryptions which have to be performed for generating/evaluating a garbled gate (which impacts the computational overhead of the protocol), and the computational assumptions under which the garbling scheme is proven secure. We included the most representative, with respect to efficiency-security trade-offs, garbling schemes into `Transputation`. All other schemes in the literature are strictly worse when it comes to either efficiency or security. In our evaluations with `Transputation`, we extend upon the work of GLNP15 (Gueron et al., 2015) to show that with a better selection of transport layer protocols, secure computation with standard assumptions can be made more efficient.

3.1 Garbling Schemes

We identify the computations and operations which have impact on the communication efficiency and explain the corresponding optimizations that we deployed (which, as we explain, do not sacrifice security). We use the garbled circuits implementations that are part of the libscapi library (Ejgenberg et al., 2012), since all the known garbled circuit optimizations have been incorporated into it. All implementations use AES-NI.

GLNP15. The first implementation is based on the garbling scheme from (Gueron et al., 2015). The main advantage of this garbling scheme is that it only makes conservative computational assumptions, i.e., it can be proven secure under the assumption that AES behaves like a pseudorandom function (PRF). Similar to all garbling schemes that we consider in `Transputation`, the garbling of linear gates (e.g., XOR) and non-linear gates (e.g., AND) is performed differently. In GLNP15, garbling an AND gate produces two ciphertexts (using the *4-2* Garbled Row Reduction technique, or *GRR* for short), while garbling an XOR gate produces one ciphertext using the *XOR-1* technique. From a computational point of view, garbling with AES key scheduling is pipelined. Circuit garbling requires four key schedules per gate while circuit evaluation requires two key schedules.

Half-Gate. The second implementation is based on the work of (Zahur et al., 2015), and uses the so called "half-gate" optimization, which in turn is compatible with the "free-XOR" optimization of (Kolesnikov and Schneider, 2008). This optimization requires a stronger computational assumption on AES, namely assuming some form of circular-security (a kind of related-key assumption). The half-gate optimization reduces the number of ciphertexts necessary to garble an AND gate from four to two, using a different approach than GLNP15. We refer to the original paper (Zahur et al., 2015) for details.

Table 1: Number of gates.

Function	AND gates	XOR gates
AES	6,800	25,124
SHA256	90,825	42,029
MinCut	999,960	2,524,920

Table 2: Garbled circuit size in Megabytes.

Assumption	AES	SHA256	MinCut
PRF	0.59	3.41	69.04
Circularity	0.21	2.77	30.52
Ideal cipher	0.21	2.77	30.52

JustGarble. The final implementation we consider was proposed in the JustGarble framework of (Bellare et al., 2013). Recall that key-scheduling is the most expensive phase when using the AES-NI, i.e., the instruction is optimized to garble large amount of data under the same key, but loses some of its efficiency when different keys have to be used all the time. In garbling schemes each gate consists of ciphertexts where different keys are used, thus the full power of the AES-NI set is not exploited. In JustGarble, AES is used as an ideal permutation "in stream cipher mode" by setting the key as a fixed constant, e.g., to encrypt message m under key k one computes $C = AES_c(k) \oplus m$ for some constant c. This usage of AES is quite non-standard, and amongst the three presented, it provides the most extreme efficiency/security trade-off.

3.2 Applications and Circuit Size

Once we have fixed the protocol and the garbling scheme, we are left with one dimension, namely, which function should we evaluate using the 2PC protocol? For garbled circuit protocols, the circuit size plays a significant role in defining the amount of data that is to be transferred over the network. The amount of data transferred from the Garbler to the Evaluator is a linear function of the circuit size. In particular, there is a difference in the price to pay (in terms of communication complexity) for linear gates vs. non-linear gates, and different garbling schemes have different coefficients for these two types of gates.

In this work, we consider three applications with circuits of three different sizes. These circuits are becoming the de-facto standards for benchmarking of MPC protocols, mostly since they represent three different orders of magnitude in circuit sizes. In particular, we benchmark `Transputation` on the circuits for AES ($\approx 10^5$ gates), SHA256 ($\approx 10^6$ gates) and MinCut ($\approx 10^7$ gates). The exact number of gates and the

distribution between AND and XOR gates for these circuits is shown in Table 1. For a particular circuit, the circuit size depends on the on the security assumptions. In Table 2, a summary of the circuit sizes in megabytes is provided.

4 TRANSPORT LAYER

In this section we describe the network characteristics which `Transputation` measures in order to optimise performance, parameters for selection of optimal transport layer protocol and the challenges of integrating transport sockets.

4.1 Transport Protocol Selection

Given the sizes of the inputs and the 2PC implementation, the transport layer of `Transputation` measures the packet loss and the latency and heuristically determines which transport protocol is optimal. The decision whether reliability and congestion control mechanisms are needed is made considering the network characteristics. `Transputation` runs continuous experiments with PING, probes the network for losses, tries different sending rates and selects a protocol that empirically produces optimal performance.

Protocol Selection based on Latency. Latency plays an important role in the choice of transport layer protocol. When the time to transmit one TCP window is longer than the round trip time (RTT), the transmission proceeds in full pipe, and is essentially similar to UDP since the congestion window does not limit the transmission. In that case if the network has packet loss TCP will be used, otherwise UDP. To determine if transmission proceeds in full pipe, `Transputation` performs the following computation: let W be the bytes in the TCP window and let t_{trans} be the transmission delay of one byte. Let RTT be the time it takes to transmit one TCP segment and receive an ACK. If $W \cdot t_{trans} > RTT$, then there is no impact of TCP congestion window on the latency since transmission proceeds in pipeline. `Transputation` measures the RTT, the window size W and the ratio $W \cdot t_{trans}$ vs. RTT and determines which transport protocol to use (i.e., window-based or to transmit in full pipe). In low latency networks, e.g., evaluations on the same LAN, congestion control is generally insignificant (typically LANs do not suffer from packet losses and have low latency). In those settings `Transputation` resorts to UDP-like protocols.

Protocol Selection based on Communication Rounds. The relevant parameters here are the number of interactions and data volume. Window based protocols, such as TCP, are not optimal for 2PC implementations with small number of interactions and large data volumes due to the fact that the transmission window of TCP increases with the number of RTTs. In TCP the window starts with one segment and increases exponentially with every received ACK. As a result, although only a few interactions are required on the application layer, they will be performed in multiple interaction rounds on the transport layer. This factor is most evident in high latency networks.

Protocol Selection based on Packet Loss. TCP performs poorly during packet loss events, even when very few packets are lost, say 1%. During packet loss, reliability should be taken care of in the user space with UDP or with SABUL.

4.2 Avoiding Packet Loss

`Transputation` avoids packet loss at the sender by adjusting the size of buffers as a function of latency, transmission rates and the data volume that needs to be transmitted. The idea is the following: the application sends packets to the transport layer and the transport layer depletes the buffer by passing the packets on to the IP layer, which subsequently transmits the packets on the wire. When the application passes the packets faster than the transport and the IP layers can process them, then the buffers will overflow and packets will be lost. `Transputation` performs adjusts the buffers, according to the computation below, to avoid packets' loss. Given (1) the differences between the transmission rate and the rate at which the data is passed on to the IP buffer (respectively transport layer) at the sender, (2) the transmission rate and the rate at which the IP buffers (and respectively transport layer) are depleted at the receiver and, (3) the input sizes and the secure computation implementation (both of which define the data volume and the rate at which it will be exchanged), `Transputation` performs a computation of the maximal amount of data that can be sent in one window.

The computation that is performed by `Transputation` is as follows. Given a receiver buffer of size B, with data arrival rate $R_{arrival}$, and the buffer depletion rate R_{read}. `Transputation` computes the maximum window size as geometric series that converges to: $L = \frac{B}{1-R_{arrival}/R_{read}}$ During the computation, the data transmitted will be limited by L bytes during each transmission window. This accounts for the data that is being read, while new data arrives, and allows to optimise the communication. This is not the same as the flow control performed by TCP, which avoids overflow at the receiver.

4.3 Transport Protocols in Transputation

We integrated into Transputation the following transport protocols: TCP, UDP and SABUL. We implemented TCP for comparison to other transport protocols. We used UDP as a benchmark for connectionless protocols on networks without losses. Examples for such networks is evaluations on LAN, or implementations which are meant to run on 5G networks, which guarantees reliability and no packet loss (Parvez et al., 2018).We integrated SABUL, as it is currently used by an increasing number of applications, and provides reasonable performance for Internet communication. Furthermore, SABUL is TCP-friendly and fair to other applications sharing the same network. New and even more efficient protocols may be developed in the future. Transputation enables easy integration of new and additional transport protocols. We describe the steps needed for integrating new protocols into Transputation in Section 5.

5 Transputation IMPLEMENTATION

The design goal of Transputation is to provide a modular design that can be extended with other secure computation protocols as well as transport layer protocols. Transputation allows secure computation researchers to focus on the protocol details without worrying about the networking aspects of the implementation. Modularity is not restricted to secure computation protocols. Transport layer protocols, ancillary to those included in the framework, can be added to the framework if required.

Abstraction. The transport layer part of Transputation is implemented as a wrapper written in C++ which can be easily plugged in with secure computation protocols. The current version uses synchronous sockets which is sufficient for our purposes. It abstracts the network functionality and removes the requirement to deal with the transport layer protocols themselves. The transport layer protocol can be set at runtime, making it easier to compare different protocols without the need for recompilation. Since most of the secure computation implementations developed in the past few years are written in C++, polymorphism in C++ are used to achieve modularity. We implement an abstract class with methods required to establish and close connections, and to send and receive data. Every transport layer protocol that is or will be implemented in our wrapper extends this class and implements these methods. This provides secure computation developers with two benefits: First, they do not need to know how to use the transport layer protocol and second, they can use new protocols that are added to the wrapper without making any change to the executable or library of the secure computation implementation. To implement a new protocol only the four methods of the abstract class are required: SetupClient, SetupServer, RecvRaw, SendRaw.

The wrapper currently supports UDP, TCP and SABUL. Since UDP does not provide reliability,it cannot be used in real-world scenarios with packet losses and where the correct order of packets cannot be guaranteed. If a dedicated network without packet loss is available, then UDP can be used. In all other cases, we recommend to use TCP or SABUL in Transputation. To choose between TCP and SABUL, we provide evaluations in Section 6.

Simplification. We have used predefined class methods to simplify common tasks. When an instance of the Transport class is created, a socket is already allocated and set up on creation (by using socket() e.g. for TCP or UDP). Then the user decides if a client which connects to a server should be created, or if a server which listens on a port and waits for incoming connections should be created. This reduces the number of lines needed as well as improves the readability and encourages users to separate program logic from network code. This is important to make the code reusable. It is also easier to test the functionality of different parts when they are separated in a modular design. The wrapper also includes two static methods, GetLatencyClient() and GetLatencyServer() to measure the latency. These methods use UDP packets to measure the RTT in milliseconds. This can be used to decide which protocol should be used. Finally, the wrapper takes care of packet sizes' byte order, which simplifies porting applications to different platforms.

Packet Handling. Transport layer protocols have contrasting methods to send data. For instance, UDP sends single packets, and hence, sending ten 100 byte packets is not an issue. However, sending packets larger than the maximal allowed packet size, limited by the underlying maximum transmission unit (MTU) of the network stack, is not possible without splitting the data into smaller chunks. This has to be done by the program that incorporates UDP, which does not provide this by itself. In contrast to UDP, TCP sends data as a stream. If the data to be sent is too large, it will be split into multiple packets by the protocol.

To solve the issue where multiple packets are received as a big chunk, the wrapper includes a Packet class which can be used to send a given amount of

data. The receiver can then, without knowing the packet size prior to receiving, receive the packet. For all protocols, that can be included in the wrapper, the data will be split into multiple packets if needed and reassembled at the receiver side. In order to tell the packets apart, the length information is added to the data so it can be interpreted by the receiving side.

Integration. The wrapper can be integrated into any secure computation protocol, by simply calling methods of the wrapper such as `Send()` or `Recv()` in the `Transport` instance. The protocols can be supplied as a string such as `udp`, `tcp`. If the wrapper is then extended with a new transport layer protocol, then the framework will be able to use this protocol without any changes. We have integrated the wrapper into libscapi. Libscapi uses external OT extension libraries which has its own network code. By incorporating our wrapper in libscapi, both—libscapi code and OT extension code—can use suitable transport layer protocols without changing the rest of the code. This is an advantage of using a common network wrapper.

6 SIMULATIONS AND EVALUATIONS

In this section we provide the simulation results which are used to understand the effect of latency, packet loss and bandwidth on the performance of secure computation protocols (Section 6.1). Then we describe realistic deployment scenarios (Section 6.2) followed by evaluation results obtained in LAN and WAN settings (Section 6.3).

6.1 Simulations

In this section, we provide the results obtained by simulating two parties performing secure computation on a single machine. The simulation results provide the benchmark for the executions in real network setups that we describe in Section 6.3 We simulate latency and packet loss using `tc qdisc` network emulator on a single Vultr instance with a 64-bit single core CPU with 2.6 GHz and 2 GB RAM. The communication takes place over the loopback interface.

Through these simulations we aim to understand the impact of latency and packet loss on secure computation protocols. We simulate latencies between 1ms to 300ms and packet losses between 0.01% and 0.05%.The latencies were chosen to represent communication between machines on the same network as well as those in different parts of the world. For instance, the round-trip time (RTT) within North America is 50ms on average, RTT between machines on either side of the Atlantic Ocean is about 100ms and machines placed in North-America and Asia or EU and Australia is about 300ms. Packet losses were chosen such that they are representative of realistic packet losses observed in networks[2]. Here we have presented the results on a 10Gbps network.

To understand the effect of latency on secure computation protocols, we run them using TCP, UDP and SABUL. We use UDP to provide the best possible runtime of the protocol if a reliable transport layer protocol is not needed. As reliable communication is required to satisfy the correctness property of secure computation in real networks, we use UDP to benchmark the runtimes achievable when the bandwidth is optimally utilized. We use two reliable transport layer protocols, TCP and SABUL, to show the bandwidth utilisation for different circuit sizes. All experiments use single thread with AES-NI.

For a small circuit such as AES, it can be observed in Figure 2 that TCP with Nagle's algorithm (Nagle, 1984) disabled performs better than SABUL. This is because for small circuits, few kilobytes of data are sent over the network while SABUL is optimized for transfer of large data transfer. For a medium-sized circuit such as SHA256, it can be observed in Figure 2 that TCP performs better than SABUL under ideal cipher assumption and circularity assumption, while under PRF assumption, the performance of SABUL is better than TCP as RTT increases. Our observation is due to a combination of reasons: the performance of SABUL is better as Bandwidth Delay Product (BDP) increases as well as when the amount of data transferred increases. For a large circuit such as MinCut, the performance of SABUL and TCP is quite different from that observed for AES and SHA256. It can be observed in Figure 2 that SABUL utilizes the available bandwidth much better than TCP. As many packets are sent from the Garbler to the Evaluator, the congestion control mechanism plays an important role in controlling the rate of packet transmission. The increase in latency affects the performance of TCP more than SABUL as SABUL uses a timer-based selective ACK instead of reacting to packet level events.

When considering packet loss, for small circuits such as AES, loss rate that we consider impacts the performance of TCP more than SABUL, as can be seen in Figure 3. For medium-size circuits such as SHA256 and large circuits such as MinCut, increase in loss rate deteriorates the performance significantly when TCP is used. On the other hand, SABUL handles packet loss better and the performance deterioration is very low. For medium and large-sized cir-

[2]http://www.verizonenterprise.com/about/network/latency/

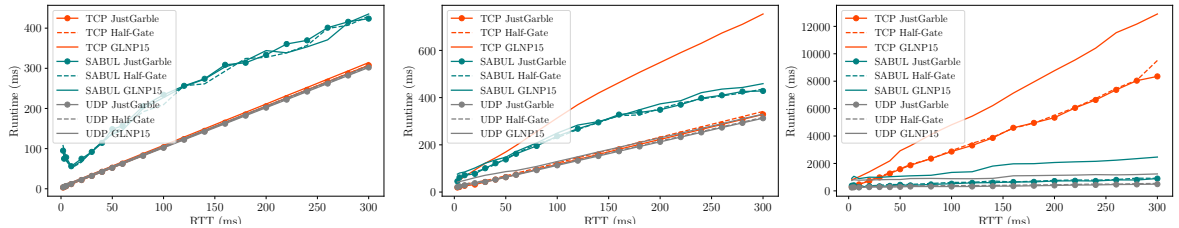

Figure 2: Effect of latency on a 10Gbps link for (a) AES (b) SHA256 (c) MinCut.

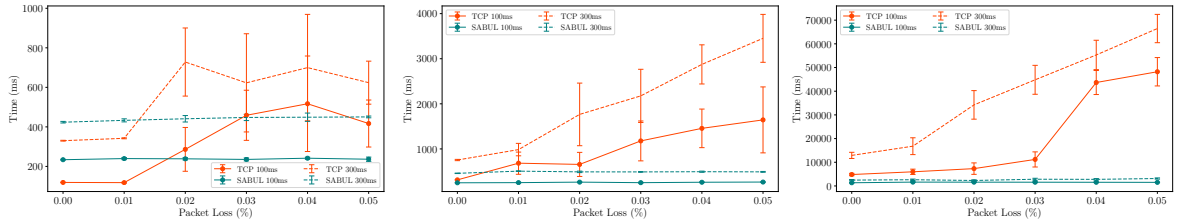

Figure 3: Effect of loss on a 10Gbps link for (a) AES (b) SHA256 (c) MinCut.

cuits on a network with at least 100ms delay, SABUL is more suitable than TCP. We have also plotted the standard deviation of the runs as the variance is non-negligible in many of the results.

6.2 Deployment Setups

For performance evaluation, two deployments were set up on our testbed environment: LAN setting and WAN setting. These settings provide two common setups of evaluation of secure computation protocols. In both setups, we use two Azure instances, each with a 64-bit Intel Xeon quadcore CPU with 2.4 GHz and 28 GB RAM.

LAN Setting. We ran the experiments on two Azure instances located in the same data centre in the EU using high-bandwidth network and low latency. The latency was 0.5 ms on a 10Gbps link. The variance was within 10%.

WAN Setting. We ran experiments on two pairs of locations with different latencies. These locations were chosen to show the behaviour of secure computation protocols with different transport layer protocols as latency increases.

EU-US: In this setting, one machine is located in the EU while the other is located in central US. The latency was 110ms and the network speed was estimated to be 1Gbps. The measured speed for a single TCP connection was 200Mbps on average. Both machines run Ubuntu 16.04. Variance of 15% was observed in this setting.

EU-AUS: In this setting, one machine is located in the EU while the other is located in south-east Australia. The latency was 300ms and the network was estimated to be 1Gbps. The measured speed for a single TCP connection was 100Mbps on average. Variance of 20% was observed in this setting for secure computation of AES and SHA256 while the variance for MinCut was 30% for TCP and 25% for SABUL.

Table 3: Experimental results for garbled circuit protocols (Runtime in ms).

Circuit	Setting	JustGarble		Half-Gate		GLNP15	
		TCP	SABUL	TCP	SABUL	TCP	SABUL
AES	LAN	**2.4**	187.1	**2.9**	191.3	**7**	202.7
	EU-US	**127.4**	403.9	**126.3**	408.3	**130.4**	444.2
	EU-AUS	**312.44**	566.84	**310.88**	580.2	**377.92**	592.5
SHA256	LAN	**13.5**	191.9	**19.9**	226.7	**30.5**	233.45
	EU-US	**146.23**	332.24	**151.99**	318.26	**266.46**	411.96
	EU-AUS	**362.53**	568.22	**394.13**	587.03	650.44	**612.43**
MinCut	LAN	**255.19**	598.2	**267.2**	740.2	**700.8**	1255.9
	EU-US	2616.59	**896.74**	2783.6	**957.6**	4911.89	**1802.25**
	EU-AUS	8204.61	**1068.07**	8693.57	**1163.14**	13805.2	**2001.27**

6.3 Experimental Evaluations

In this section we present the results obtained by using `Transputation` for the three secure computation protocols using TCP and SABUL for communication in LAN and WAN settings. All experiments are run using single thread and AES-NI.

We summarize the experimental results in Table 3. The timing in the table include the garbling time, transfer of data from the Garbler to the Evaluator and the computation of output. In LAN setting, TCP performs best for all circuit sizes. This is because, when Nagle's algorithm (Nagle, 1984) is disabled, the packets are sent as soon as they arrive at the buffer. Disabling Nagle's algorithm is advantageous in LAN setting as the communication is fast and computation consumes bulk of the time taken by the protocol.

In the WAN setting, when the 2PC protocol is run between Azure instances in EU and US, the circuit size begins to influence the performance. For AES and SHA256, TCP is still more efficient than SABUL but the tide tilts for MinCut. MinCut using SABUL is $2.7-3\times$ faster than TCP. In fact, secure computation of MinCut under PRF assumption using SABUL is more efficient than under ideal cipher assumption or circularity assumption using TCP.

In the WAN setting, when the secure computation protocol is run between Azure instances in EU and Australia, the influence of latency on secure computation of large circuits becomes much more evident. Secure computation of SHA256 under PRF assumption (with 223,679 ciphertexts) using SABUL is only a little faster than using TCP. Secure computation of MinCut using SABUL is $7-8\times$ faster than using TCP. This is significantly more than the improvements that can be expected from secure computation protocol improvements (Zahur et al., 2015).

7 RELATED WORK

The landscape of secure multiparty computation protocols is broad and diverse, and different protocols exist for different number of parties, adversarial models, corruption thresholds, etc. In this paper we have chosen to focus on the most natural case of secure two-party computation, and we leave the investigation of the multiparty case as future work.

Secure Computation. Secure 2PC protocols have been extensively studied since mid-1980s when the first feasibility results were provided by Yao (Yao, 1986) and Goldreich, Micali and Wigderson (GMW) (Goldreich et al., 1987). These seminal results show that it is possible to evaluate any function in a secure way, that is, two parties P_1, P_2 with inputs x, y can jointly evaluate some function f (expressed as a Boolean circuit) on their inputs in such a way that both parties learn the desired output $z = f(x,y)$ *and nothing else* about the input of the other party.

A long line of works have tried to improve the efficiency of garbled circuits. The original protocol by Yao (Yao, 1986) requires a transfer of 4 ciphertexts per Boolean gate in the circuit, and requires 4 decryptions for the evaluation of a garbled gate. The number of decryptions per gate in the evaluation phase was reduced to 1 due to the *point-and-permute* strategy of (Beaver et al., 1990), which also reduces the size of the ciphertexts by a factor 2 (approximately). The main measure of *communication complexity* of garbling schemes is the number of ciphertexts which are transmitted per Boolean gate. This was first reduced in (Naor et al., 1999) using the *garbled row-reduction technique (GRR)* to 3 ciphertexts per gate (the so called 4-3 GRR technique) and to 2 ciphertexts per Boolean gate in (Pinkas et al., 2009) (e.g., 4-2 GRR). Thanks to the *free-XOR* technique (Kolesnikov and Schneider, 2008), it is not necessary to transfer any ciphertexts for XOR (or other linear) gates. Unfortunately the *free-XOR* technique was incompatible with the more advanced 4-2 GRR technique. The two techniques were finally combined thanks to the *half-gate* optimization, which combines the benefit of *free-XOR* (no ciphertexts for XOR gates) with the advanced 4-2 GRR technique (only 2 ciphertexts per AND gate). Unfortunately, the free-XOR technique is only secure under non-standard cryptographic assumption (Choi et al., 2012) and, in particular, it requires some form of "circular security" assumption. Using an even stronger assumption, i.e., *fixed-key* AES behaves like a *random permutation*, faster garbling schemes were proposed in (Bellare et al., 2013).

While efficiency is a crucial aspect in 2PC, some have questioned whether it is wise to make protocols efficient at the cost of strong assumptions. In particular, we note that a conservative approach is usually adopted by the industry as it is difficult to change protocols if vulnerabilities are discovered after deployment. Therefore, (Gueron et al., 2015) provided novel constructions of garbled circuits that can be proven secure using standard assumptions only and that require 2 ciphertexts for AND gates (4-2 GRR) and 1 ciphertext for XOR gates (XOR-1). (Gueron et al., 2015) conclude that the price to pay for the stronger security guarantees in practice is much less than it is in theory. Our evaluations in some sense confirm and strengthen the conclusions of (Gueron et al., 2015): our experiments show that the choice of the right transport protocol has a much higher impact on overall efficiency than gambling on security by using non-standard assumption. In particular (when evaluating large circuits over WAN), using SABUL plus standard assumptions is $4\times$ faster than using TCP plus non-standard assumptions.

Secure Computation Frameworks. Since the implementation of Fairplay framework (Malkhi et al., 2004) in 2004, various frameworks have been developed for secure 2PC (Huang et al., 2011; Ejgenberg et al., 2012; Demmler et al., 2015). Currently, a garbled circuit framework with security against passive and active adversaries is provided in libscapi (Ejgenberg et al., 2012) library with the latest optimizations. All the previous frameworks focus on secure computation and use standard TCP socket provided by the operating system. The recent work of (Hastings et al., 2019) surveys general purpose compilers for secure

computation, which provide high-level abstractions to describe functions in an intermediate representation (such as a circuit). While (Hastings et al., 2019) focuses on compilers, we focus on efficiency of protocol execution. Hence, the two works address orthogonal problems. Another recent work is (Barak et al., 2018), which considers the possibility of providing MPC as a service where users use the platform to run protocols. They provide an environment where users can participate in a low-bandwidth MPC protocol using web-browser or an app on the phone. While they focus on low-bandwidth MPC protocols, we focus on high-bandwidth constant round 2PC protocols.

Related work shows that significant progress has been made in computation complexity of 2PC implementations. When evaluated on a single host the implementations produce good performance. However, on real networks with other traffic and concurrent processes, latency and packet loss, the efficiency collapses and implementations incur prohibitive latency. In particular we note that no previous work has considered how to improve secure computation by addressing the issue of transport layer performances, as addressed in this work.

8 CONCLUSION

Our work demonstrates that, contrary to folklore belief, bandwidth is *not* the bottleneck in performance of secure computation. The performance of 2PC implementations can be significantly improved if other transport protocols are used instead of the standard TCP sockets. Our evaluations demonstrate performance improvements for 2PC even with general purpose transport layer protocols, e.g., SABUL is 8× more efficient than TCP for the same task and 4 times more efficient than TCP when comparing 2PC with standard assumptions over SABUL vs 2PC with non-standard assumptions over TCP.

Support of Multiple Protocols. To enable 2PC developers to benefit from the wide range of existing protocols, we have developed `Transputation`. Users can setup `Transputation` locally or can use our installation (with preconfigured 2PC implementations and transport protocols) which can be accessed online. `Transputation` is running on the testbed with representative network setups for evaluation of secure computation implementations.

Evaluation of 2PC in Realistic Setups. Although evaluations on a single host provides practical results, they "break" when run in real Internet networks. We setup a testbed, which automates setup and configurations, to evaluate 2PC implementations and to allow developers to run evaluations without having to dive into complicated setups, renting remote machines and running network measurements.

ACKNOWLEDGMENT

This work has been co-funded by: the Concordium Blockhain Research Center, Aarhus University, Denmark; the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC); the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC); the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE; the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation): GRK 2050/251805230 and SFB 1119/236615297.

REFERENCES

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. (2011). Data center tcp (dctcp). *ACM SIGCOMM computer communication review*, 41(4):63–74.

Archer, D. W., Bogdanov, D., Lindell, Y., Kamm, L., Nielsen, K., Pagter, J. I., Smart, N. P., and Wright, R. N. (2018). From keys to databases - real-world applications of secure multi-party computation. *Comput. J.*, 61(12):1749–1771.

Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. (2013). More efficient oblivious transfer and extensions for faster secure computation. In *ACM Conference on Computer and Communications Security*, pages 535–548. ACM.

Barak, A., Hirt, M., Koskas, L., and Lindell, Y. (2018). An end-to-end system for large scale P2P mpc-as-a-service and low-bandwidth MPC for weak participants. In *CCS*, pages 695–712. ACM.

Beaver, D., Micali, S., and Rogaway, P. (1990). The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513. ACM.

Bellare, M., Hoang, V. T., Keelveedhi, S., and Rogaway, P. (2013). Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society.

Bellare, M., Hoang, V. T., and Rogaway, P. (2012). Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, pages 784–796. ACM.

Bogdanov, D., Jõemets, M., Siim, S., and Vaht, M. (2015). How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party

computation. In *Financial Cryptography*, volume 8975 of *Lecture Notes in Computer Science*, pages 227–234. Springer.

Bogetoft, P., Christensen, D. L., Damgård, I., Geisler, M., Jakobsen, T. P., Krøigaard, M., Nielsen, J. D., Nielsen, J. B., Nielsen, K., Pagter, J., Schwartzbach, M. I., and Toft, T. (2009). Secure multiparty computation goes live. In *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer.

Caini, C. and Firrincieli, R. (2004). Tcp hybla: a tcp enhancement for heterogeneous networks. *International journal of satellite communications and networking*, 22(5):547–566.

Choi, S. G., Katz, J., Kumaresan, R., and Zhou, H. (2012). On the security of the "free-xor" technique. In *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 39–53. Springer.

Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., and Bowman, M. (2003). Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12.

Demmler, D., Schneider, T., and Zohner, M. (2015). ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS*. The Internet Society. Code: https://github.com/encryptogroup/ABY.

Ejgenberg, Y., Farbstein, M., Levy, M., and Lindell, Y. (2012). SCAPI: the secure computation application programming interface. *IACR Cryptology ePrint Archive*, 2012:629. Code: https://github.com/cryptobiu/libscapi.

Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM.

Gu, Y. and Grossman, R. L. (2008). Udtv4: Improvements in performance and usability. In *GridNets*, volume 2 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 9–23. Springer.

Gueron, S., Lindell, Y., Nof, A., and Pinkas, B. (2015). Fast garbling of circuits under standard assumptions. In *ACM Conference on Computer and Communications Security*, pages 567–578. ACM.

Ha, S., Rhee, I., and Xu, L. (2008). Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74.

Halevi, S. (2018). Advanced cryptography: Promise and challenges. In *ACM Conference on Computer and Communications Security*, page 647. ACM.

Hastings, M., Hemenway, B., Noble, D., and Zdancewic, S. (2019). Sok: General purpose compilers for secure multi-party computation. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1220–1237. IEEE.

Hemenway, B., Lu, S., Ostrovsky, R., and IV, W. W. (2016). High-precision secure computation of satellite collision probabilities. In *SCN*, volume 9841 of *Lecture Notes in Computer Science*, pages 169–187. Springer.

Huang, Y., Evans, D., Katz, J., and Malka, L. (2011). Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*. USENIX Association.

Kolesnikov, V. and Schneider, T. (2008). Improved garbled circuit: Free XOR gates and applications. In *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer.

Kreuter, B. (2017). Secure multiparty computation at google. In *Real World Crypto Conference (RWC)*.

Malkhi, D., Nisan, N., Pinkas, B., and Sella, Y. (2004). Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302. USENIX.

Nagle, J. (1984). Congestion control in ip/tcp internetworks. *RFC 896*.

Naor, M. and Pinkas, B. (2001). Efficient oblivious transfer protocols. In *SODA*, pages 448–457. ACM/SIAM.

Naor, M., Pinkas, B., and Sumner, R. (1999). Privacy preserving auctions and mechanism design. In *EC*, pages 129–139.

Nielsen, J. B., Schneider, T., and Trifiletti, R. (2017). Constant round maliciously secure 2pc with function-independent preprocessing using LEGO. In *NDSS*. The Internet Society.

Obata, H., Tamehiro, K., and Ishida, K. (2011). Experimental evaluation of tcp-star for satellite internet over winds. In *2011 Tenth International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 605–610. IEEE.

Parvez, I., Rahmati, A., Guvenc, I., Sarwat, A. I., and Dai, H. (2018). A survey on low latency towards 5g: Ran, core network and caching solutions. *IEEE Communications Surveys & Tutorials*, 20(4):3098–3130.

Pinkas, B., Schneider, T., Smart, N. P., and Williams, S. C. (2009). Secure two-party computation is practical. In *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer.

Pinkas, B., Schneider, T., and Zohner, M. (2018). Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35.

Schneider, T. and Zohner, M. (2013). GMW vs. yao? efficient secure two-party computation with low depth circuits. In *Financial Cryptography*, volume 7859 of *Lecture Notes in Computer Science*, pages 275–292. Springer.

Wang, X., Ranellucci, S., and Katz, J. (2017). Authenticated garbling and efficient maliciously secure two-party computation. In *CCS*, pages 21–37. ACM.

Wu, H., Feng, Z., Guo, C., and Zhang, Y. (2013). Ictcp: Incast congestion control for tcp in data-center networks. *IEEE/ACM Transactions on Networking (ToN)*, 21(2):345–358.

Yao, A. C. (1986). How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society.

Zahur, S., Rosulek, M., and Evans, D. (2015). Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer.