

On the Performance of a Trustworthy Remote Entity in Comparison to Secure Multi-Party Computation

Robin Ankele and Andrew Simpson

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
{robin.ankele, andrew.simpson}@cs.ox.ac.uk

Abstract—Novel trusted hardware extensions such as Intel’s SGX enable user-space applications to be protected against potentially malicious operating systems. Moreover, SGX supports strong attestation guarantees, whereby remote parties can be convinced of the trustworthy nature of the executing user-space application. These developments are particularly interesting in the context of large-scale privacy-preserving data mining. In a typical data mining scenario, mutually distrustful parties have to share potentially sensitive data with an untrusted server, which in turn computes a data mining operation and returns the result to the clients. Generally, such collaborative tasks are referred to as secure multi-party computation (MPC) problems. Privacy-preserving distributed data mining has the additional requirement of (output) privacy preservation (which typically is achieved by the addition of random noise to the function output); additionally, it limits the general purpose functionality to distinct data mining operations. To solve these problems in a scalable and efficient manner, the concept of a Trustworthy Remote Entity (TRE) was recently introduced. We report upon the performance of a SGX-based TRE and compare our results to popular secure MPC frameworks. Due to limitations of the MPC frameworks, we benchmarked only simple operations (and argue that more complex data mining operations can be established by composing several basic operations). We consider both a two-party setting (where we iterate over the number of operations) and a multi-party setting (where we iterate over the number of participants).

1. Introduction

Computational tasks are increasingly performed at a large scale and in distributed settings, collecting sensitive information pertaining to many individuals. The collection and processing of voluminous information is useless unless meaningful information can be extracted from it. Data mining or knowledge discovery provides algorithms and operations in order to analyse and illustrate such information. In addition, privacy-preserving data mining enables multiple parties to collaboratively perform a data mining task on aggregations of their data, without revealing the private contents of their datasets.

Generally, such collaborative tasks are referred to as secure multi-party computation (MPC). In this context, a

secure multi-party protocol allows a set of parties to collaboratively evaluate a function on their private inputs, without revealing these private inputs to the other participants in the protocol. This can be any generic function — even differing for each participant. MPC protocols can be based on various cryptographic primitives, such as garbled circuits, oblivious transfer, secret sharing or homomorphic encryption. Overall, a MPC protocol is either secure against passive (*i.e.* semi-honest) adversaries, or, considering a stronger threat model, secure against active (*i.e.* malicious) adversaries.

While secure multi-party protocols do not consider output privacy, privacy-preserving distributed data mining (PPDDM) does. More specifically, the problem of PPDDM strengthens the definition of MPC by requiring output privacy preservation; additionally, it limits the general purpose functionality to distinct data mining operations. Algorithms and protocols implementing PPDDM tasks typically involve the combination and convergence of MPC protocols and privacy-preserving operations. Output privacy can be achieved by adding noise (according to a predefined probability distribution) to the functional output, by distorting or submerging certain values, or by destroying the linking relationship between the input values and the participants.

To support many participants (or many operations) in large-scale and distributed contexts, these implementations must be designed to be scalable and efficient. Over the last decade, secure multi-party protocols have evolved from theoretical endeavours to practical solutions. Moreover, recently, trusted hardware extensions such as Intel’s Software Guard Extensions (SGX)¹ have been released. SGX enables a user-space application to be run in an isolated execution area (so-called *enclaves*) and protects the process from interference of higher priority operations (such as a malicious operating system, or other malicious applications). Furthermore, SGX offers strong attestation assurances to establish a trust relationship with remote parties. Evolving from these developments, Paverd [1] proposed a Trustworthy Remote Entity and Ankele *et al.* [2] suggested its application to PPDDM. A TRE acts as a verifiable trusted third party (TTP), similar to an idealised TTP from the MPC domain. The security of a TRE is based on the strong attestation assurances from, and, indeed, on the security model of,

1. <https://software.intel.com/en-us/sgx>

SGX. Our results show that the TRE approach is more efficient and scalable in large-scale distributed environments.

In a typical data mining scenario, multiple mutually distrustful parties have to share their data with an untrusted server, which computes and returns a result. We consider such a scenario in this paper. Each participant (and the server) is represented as a user-space process on the same untrusted computational unit². To provide privacy, the communication between the processes is run by a multi-party protocol. In the case of the TRE, the processing of the sensitive data is performed in the isolated execution environment.

We report upon the performance of a SGX-based TRE and compare our results to a selected set of popular two-party and multi-party frameworks. Specifically, we evaluate several basic data mining operations — in the two-party case we increase the number of operations, and in the multi-party case we increase the number of participants. Due to limitations of the MPC frameworks, we are forced to consider only basic operations; however, we argue that these can be combined to support higher level, more complex use cases. More generally, we organise the benchmarked operation into the following categories: arithmetic, Boolean, comparison and composite operations.

We acknowledge that our benchmarking has been undertaken in doesn’t cover all possible scenarios in which MPC might be applied (and, consequently, might leave one with the mistaken impression that MPC is a limited approach or that it achieves only limited performance). However, our concern in this paper is techniques that are capable of solving general MPC problems in large-scale and distributed environments. Thus, only a few MPC frameworks meet our requirements (some of which might be considered ‘old’) — and it is these that we base our benchmarking on. Further, the operations considered are due to restrictions imposed by what those frameworks can support. Our results should be seen through that perspective.

2. Preliminaries

2.1. Secure Multi-Party Computation

The secure multi-party computation problem is defined as follows. A set of parties, P_1, \dots, P_n , each with their private input, x_1, \dots, x_n , jointly compute a function $f(x_1, \dots, x_n) = y$. The collaborative function f can be any function, and possibly a different function for each of the participating parties. An algorithm is said to be secure in the multi-party setting [3] if it fulfills the properties of *privacy*, *correctness*, *independence of inputs*, *fairness*, and *guaranteed output delivery*. Such an algorithm can achieve different levels of security, depending on the behaviour of a static or an adaptive adversary, namely *semi-honest* (i.e. *passive*), *covert*, or *malicious* (i.e. *active*) security.

2. We process the multi-party protocols on the same computational unit, to perform our benchmarking, and to be independent of any occurring, noisy network traffic. We note that each protocol is capable of running over multiple distributed parties.

A *semi-honest* adversary does not deviate from the given protocol, however she tries to gain knowledge about the other parties’ inputs by participating in the protocol execution. A subtle, more involved adversary type is that of a *covert* adversary, which deviates from the protocol only if she is confident that she will not get caught. The strongest type of adversary represents a *malicious* adversary, who arbitrarily deviates from the protocol to block, modify or replay any message or synthesise falsified messages.

In recent years, secure multi-party computation has shifted from being a theoretical endeavour towards becoming practical and deployable in real-world applications. This started in the early 1980s with the seminal work of Yao [4] on garbled circuits, and the consequential protocols GMW [5] and BMR [6], similarly based on Boolean circuits, as well as the BGW [7] protocol, based on arithmetic circuits. In order to achieve privacy in these protocols, Yao and GMW utilise oblivious transfer [8], [9], while BMR and BGW make use of secret sharing [10], [11]. Modern protocols combine previous techniques or take advantage of other cryptographic concepts such as homomorphic encryption [12], [13], commitment schemes [14], or zero-knowledge proofs [13].

2.2. Software Attestation

Before a party decides to outsource its sensitive data values to a remote entity, a *trust relationship* needs to be established with such an entity. The trust establishment assures that the remote entity behaves in a predefined manner and performs the agreed operations on the data. A principal mechanism to establish such a trust relationship is *software attestation*. The entity that provides a statement or some sort of information of trust is called a *prover* and the entity verifying the statement is called a *verifier*.

The information itself is, in most cases, some sort of *measurement*. Typically, such a measurement is computed by using a cryptographic hash function and computed over the code and data pages of the algorithm and operating system of the remote entity (i.e. the trusted computing base). During the launch (or in special cases also during the runtime) of the remote entity each code page loaded into operating memory is measured and stored into reserved registers or memory locations by extending the current value (by hashing the current value and the new page).

To guarantee the validity of an attestation message, the verifying entity must trust the *root of trust for measurement* and the *root of trust for reporting* of the proving entity. The former element (or function) ensures the integrity of the measurement process and the latter element ensures integrity and validity of the reporting process. Generally, to observe a valid attestation and furthermore a successful establishment of a trust relationship, the following properties must be fulfilled.

Authenticity. The verifying entity must be able to unambiguously determine that the measurements were provided from the proving entity.

TABLE 1: Selection of Protocols for Two-Party and Multi-Party Computation.

Protocol	Type	Security	Cryptographic Primitive	Reference
Fairplay	two-party	malicious	Boolean circuits	[9]
Tasty	two-party	semi-honest	oblivious transfer garbled circuits* homomorphic enc.	[15]
FairplayMP	multi-party	semi-honest	Boolean circuits	[16]
VIFF	multi-party	malicious	arithmetic circuits secret sharing (Shamir)	[17]

* arithmetic and Boolean circuits.

Currentness. The verifying entity must be able to unambiguously determine that the measurements represent the current state of the proving entity.

Public-key signature schemes (e.g. RSA or ECC) are generally used in order to provide authenticity in attestation protocols. However, this may lead to privacy problems if several verifiers collude and track the behaviour of a prover. In order to prevent such cases, the use of a *Privacy Certification Authority*, *zero-knowledge proofs*, or *commitments* are recommended. Furthermore, to prevent the replay of older attestation messages, timestamps or nonces can be applied.

Attestation can be performed locally on the same platform, which is referred to as *local attestation*, or globally to remote entities, which is referred to as *remote attestation*. The performance of an attestation protocol plays an important role, especially in remote attestation. Driving performance factors are latency (i.e. time of a single attestation) and scalability (i.e. rate of verifying parties in a give time interval) of the attestation protocol.

2.3. Intel’s Software Guard Extensions

In 2013, Intel announced a new security technology, Intel Software Guard Extensions (SGX), which comprises a novel microprocessor architecture instruction set extension available with the Skylake micro-architecture. The following features are supported by a SGX-enabled remote party (equipped with an Intel CPU):

- 1) The remote party cannot alter the code or data pages of the program being executed, and therefore provides *integrity*.
- 2) The remote party can process encrypted code and data pages, which are only decrypted inside the CPU (and its components), and therefore provides *confidentiality* of the program and remains inaccessible to other software on the remote party (e.g. the operating system).

SGX achieves the trusted computing functionality by deploying so-called secure enclaves — small pieces of code and data pages within a more complex user-space application that is isolated from the other software running on the remote party. SGX enclaves run in the same process as the host application. Access to code or data in the enclave must go through an ECALL and calls from the enclave to the outside, handled by the operating system, must go through an OCALL. With SGX it is possible to deploy one or more secure enclaves in the same application, with each utilising their own isolated memory pages.

Summarising, SGX ensures: hardware-enforced isolation of code and data memory pages; encryption of blobs bound to the enclave identity (i.e. MRENCLAVE), or encryption of blobs bound to the enclave’s sealing identity (i.e. MRSIGNER); and privacy-preserving local and remote attestation based on a group based signatures scheme called Enhanced Privacy Identifier (EPID).

3. Secure Two-Party and Multi-Party Protocols

Several implementations for secure two-party and multi-party computation have been published in the last decade. These include the two-party protocols Fairplay [9], Tasty [15] and MightBeEvil [18], as well as the multi-party protocols FairplayMP [16], VIFF [17], Sharemind [19], SEPIA [20], SPDZ [13] and SCAPI [21]. For our analysis we have selected two protocols from each category, where the selected protocols (introduced in more detail below) offer high level languages or APIs to implement secure generic functions. Moreover, we rejected some of the above mentioned frameworks, because they are implemented in ways that support only very limited special use cases, or require coding of functionality at a circuit level. A summary of the selected protocols is given in Table 1.

3.1. Two-Party Protocols

3.1.1. Fairplay. Fairplay [9] was introduced in 2004 and represents one of the first automated frameworks for secure function evaluation. It is based on Boolean circuits and utilises oblivious transfer for secure communication between two parties. In Fairplay a collaborative function is defined in a high-level procedural definition language called SFDL. Using the Fairplay compiler the function is transferred into a one-pass Boolean circuit structure called SHDL and can be evaluated using so called Bob/Alice programs. Fairplay achieves *active* security using the *cut-and-choose* method [22].

3.1.2. Tasty. In 2010, Tasty [15] was introduced, which improves upon Fairplay by allowing the user to generate protocols based on garbled circuits (i.e. Boolean and arithmetic) and homomorphic encryption, as well as combinations of both. The Tasty framework comprises: a compiler for a domain-specific language called TastyL; server and client programs for the secure function evaluation; and tools for testing, benchmarking and post-processing of any protocol implementation. Furthermore, Tasty supports the evaluation

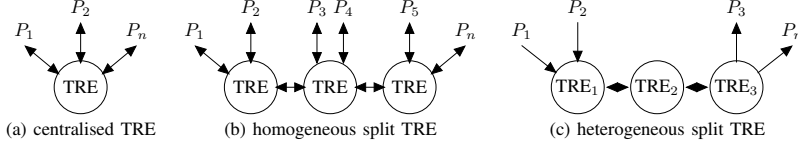


Figure 1: Different architectural design models of a Trustworthy Remote Entity.

of circuits in SHDL layout, generated by the Fairplay compiler. Tasty has four predefined security levels (80-, 96-, 112- and 128-bit) for *passive* security based on elliptic curves.

3.2. Multi-Party Protocols

3.2.1. FairplayMP. FairplayMP [16] (introduced in 2008) improves upon Fairplay by introducing a multi-party version and several other optimisations. It is based on the BMR protocol for Boolean circuits, which supports a constant number of communication rounds and the BGW protocol for the purpose of constructing gate tables. FairplayMP comprises a compiler that supports the SFDL 2.0 language — an extension of the two-party Fairplay’s SFDL — and a cryptographic engine for the evaluation of secure multi-party protocols. Protocols designed in FairplayMP are based on input, computational and result players. It focuses on protocols which provide security against *semi-honest* adversaries.

3.2.2. VIFF. The Virtual Ideal Functionality Framework [17] (introduced in 2009) is a software framework to design secure applications in the multi-party setting — it deploys asynchronous protocols for general multi-party computation based on arithmetic circuits and Shamir’s secret sharing [10]. VIFF allows for parallelisation of primitive operations without multi-threading. The asynchronous protocols are split into an offline pre-computation and input phase generating multiplication triples and an online computation phase evaluating these multiplication triples. Protocols generated with VIFF are perfectly secure against adaptive malicious adversaries corrupting fewer than $\frac{n}{3}$ players.

4. Trustworthy Remote Entity

In 2014 Paverd [1] introduced the concept of a Trustworthy Remote Entity (TRE) as a single, centralised and general purpose entity, which serves as an intermediary between participating communication parties. Paverd’s TRE implementation, based on Intel’s Trusted Execution extensions (Intel TXT) and a Trusted Platform Module (TPM), was applied to enhance the communication privacy in the smart grid. Paverd’s contribution includes the finite state attestation protocol, which was specifically designed to address the security and performance requirements of a TRE. The general idea behind this protocol is that it reaches a final state in the measurement process, which the TRE doesn’t leave voluntarily (only by resetting the computational unit) and is thereby able to convince multiple verifiers with the assertions of a single prover.

A TRE behaves in the sense of a trusted third party (TTP), but, contrary to a TTP — where users are required

to blindly trust all actions performed by the TTP — a TRE can verify its trustworthy nature. In other words, a TRE is a verifiable TTP that is able to prove its trustworthiness to other participants via software attestation. Ankele *et al.* [2] and Küçük *et al.* [23] established a SGX-based TRE for application to MPC and PPDDM. Using SGX as the underlying hardware security primitive, the SGX-based TRE solution can benefit from the various SGX features such as isolated execution of code and memory pages, encrypted storage outside of the immediate CPU package, and the software attestation features of SGX. Further, the TRE can be executed directly on the main processor core, which in turn offers highly scalable and efficient execution while reducing the trusted computing base of the overall system.

A TRE is not bound by any architectural design choices and can be implemented by any of the design models given in Figure 1. In particular, these comprise either a single *centralised* unit or multiple distributed components. We focus only on the single centralised model, where each party verifies the TRE attestation assertions and then shares their input with the TRE, which computes and returns the result. In the distributed case, the TRE may be deployed in a *homogeneous* split architecture (*i.e.* every component performs the same operations) or in a *heterogeneous* split architecture (*i.e.* every component performs a different operation, but operate together to achieve a global goal). We do not consider these latter scenarios in this paper.

The security assumptions of the SGX-based TRE are based on those of the SGX model. In the context of secure multi-party protocols, participants establish a secure channel following their trust establishment through the inter platform (remote) attestation process. During enclave initialisation, the enclave identity register MRENCLAVE and the platform-sealing register MRSIGNER are initialised and extended by SHA-256 hashes. The EREPORT command issues a signed report over the two identity registers, some user data (which can include parameters for key agreement) and other metadata, signed by an attestation identity key. A special *quoting enclave* locally attests the application enclave and the EPID group signature scheme is used to provide privacy for the attestation. Following the establishment of a secure channel between the participants and the TRE, each party can share their inputs with the TRE (via the authenticated encryption secure channel), which in turn forwards it to the application enclave. As previously discussed, every code and data page within the enclave is ensured to be confidentiality-, integrity- and freshness-protected by hardware enforcement. The application enclave processes its function on the shared inputs and returns the result via the same secure channel.

TABLE 2: Basic data mining operations used in our benchmarking.

	arithmetic	Boolean	comparison	composite
2-party	add mult	and xor	max min equality	count
n-party	sum product	and xor	max min	count average

5. Performance Evaluation

5.1. Scenario

To ease the performance benchmarking, we consider the parties to be local (*i.e.* executing on the same computational unit). In particular, parties participating in the multi-party protocol are represented as a user-space process running on the same untrusted party — an appealing feature of cloud computing. Nevertheless, in multi-party computation, in general, the participants are distributed over several parties and compute the functionality by sharing information with each other. However, this implies a communication overhead and, most of all, every participant needs to reserve some of their resources for the computation of the function. Considering IoT devices, which are typically resource-limited, outsourcing the computation to a cloud-based high-end server — which computes the functionality and returns the result — may result in a more scalable and efficient operation. In our scenario, we consider that each participant is invoking a user-space process on the untrusted server. In order to collaboratively compute the functionality, these mutually distrustful parties interact in a multi-party protocol. For the sake of simplifying the benchmarking, each user-space process is automatically invoked from our benchmarking suite and provided with a random input value (within a range of 4 bits). The server functionality is similarly represented by a user-space process. A socket implementation is used for communication between server and participant processes. In general, the performance measurements may also be applied in the distributed setting; however, involving remote parties, we need to consider a communication overhead over the network. Nevertheless, we argue that this should be a constant factor and should be irrelevant for our comparison.

5.2. Threat Model

To better illustrate our scenario we may consider two types of adversary: a malicious cloud provider and a malicious outsider. A malicious cloud provider would have access to both hardware and software of the computational device (*i.e.* cloud server), while a malicious outsider may have access only to the software. In general, attacks against *availability* (and therefore attacks against the *guaranteed output delivery* property) may be unavoidable, since an adversary controlling the operating system (and, thus, the scheduling of the processes) may always exclude the processes necessary to run the multi-party protocol. Attacks

against *privacy* may be avoided by running the previously introduced two-party or multi-party protocols. In the case of the SGX-based TRE, a rogue operating system or any other malicious application is not able to disturb the protocol execution without detection.

5.3. Performance Evaluation

We have designed a benchmarking suite in `python`, which automatically invokes the multi-party protocol and supplies a random input for every party. In particular, each participant is represented by a user-space process, running on the same computational unit (*i.e.* local setting). This follows exactly the scenario of Section 5.1. Table 6 shows, in pseudo-code, the implementation strategy of our testing.

We evaluated the performance in the context of basic data mining operations as per Table 2. These operations can be classified according to their operational categories (arithmetic, Boolean, comparison or composite) and are dependent on their applicability to two-party computation or multi-party computation. For two-party computation, we increased the number of operations, while, for multi-party computation, we increased the number of participants.

Various limitations of some multi-party protocol implementations (*e.g.* out-of-memory errors due to large Boolean circuits, out-of-time exceptions⁴, or not supported or not implemented operations) forced us to bound the number of operations/participants to a lower bound than expected. We limited the range of the input and output values (and therefore the size of protocol-internal values) to 4-bit values. In this context, we were forced to evaluate only basic data mining operations; however, higher level use cases may be built by the combination of basic operations. For example, an auction scheme consists of `sum` and `max` operations, a voting scheme involves `compare`, `count` and `max`, and set intersections comprise `compare` and `count`.

To set a base point for the comparison, we developed a plain implementation for the two-party and the multi-party cases. Our TRE implementation builds from this plain implementation by performing the sensitive operations within secure enclaves and establishing trust through remote attestation. All tests were performed on a Dell OptiPlex 7040 equipped with a SGX-enabled 64-bit Intel Core i7-6700 (3.4 GHz) quad-core processor and 32 GB DDR4 RAM, running Ubuntu 14.04. To utilise SGX, we used the SGX SDK in version 1.5. Further, we used `java` in version 1.8.0 101 and `python` in version 2.7.6. As previously mentioned, our benchmarking suite is written in `python` and utilises

3. The results of the other basic data mining operations are similar (within the standard deviation of our measurements).

4. We limited the pre-computation and online phase to, at most, 10 minutes.

TABLE 3: Results for the benchmarking of the two-party protocols for the operation add^3 . Results are given in seconds.

Name/Ops	10	100	200	300	500	700	1000	10k	100k
Plain	0.021	0.020	0.021	0.021	0.021	0.021	0.021	0.029	0.28
TRE	0.056	0.056	0.057	0.057	0.058	0.058	0.058	0.069	0.30
Fairplay	0.50	1.35	-*	-	-	-	-	-	-
Tasty	0.43	3.45	9.28	18.25	44.97	84.34	169.49	-**	-

* Results in an out-of-memory error for 200+ operations.

** Exceeds timing limitation by our benchmarking framework.

the `time.time()` function as the time source for the measurements. To avoid any unnecessary context switches and other artefacts, each result consists of the average of 10 single measurements.

Every two-party or multi-party protocol has two phases: an offline (pre-computation) phase and an online phase. The pre-computation phase comprises key generation and key exchanges between the input parties, generation of secret shares, or generation of garbled circuits. Keys might be exchanged by the means of oblivious transfer. In the case of SGX, the trust establishment via remote attestation might be included in the pre-computation phase, however, attestation needs to be repeated if the executing function at any party changes. To avoid this, we include the trust establishment via remote attestation in the online phase. In the online phase, the desired function is evaluated, for example, by evaluating the previously exchanged garbled circuit. When finished, the results are sent to the output party.

We evaluated various basic data mining operations. The results for the `add` operation in the two-party case are given in Table 3 and illustrated in Figure 2. In the case of the two-party evaluations, we fixed the number of participants to two and increased the number of operations. The results for the `sum` operation in the multi-party case are given in Table 4 and illustrated in Figure 3. In this context, we fixed the number of operations per party to one operation, however, we increased the number of participants. The results of the other basic data mining operations are similar (within the standard deviation of our measurements). This implies that the evaluated functionality is independent (in the sense of execution time) of the core of the two-party or multi-party protocol, i.e. the evaluated functionality of the protocol does not influence the whole execution time of the protocol in a measurable manner.

From the results of the two-party evaluation, we can conclude that the plain and our TRE implementation is nearly constant up to 10,000 operations. The performance drawbacks of the TRE are mostly influenced by the remote attestation (approximately 50ms [23]). Both two-party protocols (Fairplay and Tasty) significantly sustain performance losses if the number of operations is increased. Fairplay is only scalable to run 4-bit operations up to a quantity of around 100, before the pre-computed circuits results in an out-of-memory exception. The results are similar in the multi-party case. While the plain and our TRE implementation (threaded implementations) are increasing slightly due to the number of rising participants, the multi-party proto-

5. The results of the other basic data mining operations are similar (within the standard deviation of our measurements).

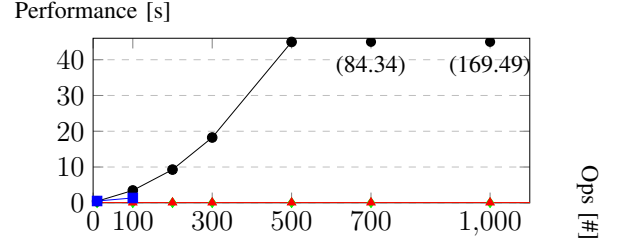


Figure 2: Comparison in the two-party setting for the operation `add`. Included two-party protocols: Plain(—♦—), TRE(—▲—), Fairplay(—■—) and Tasty(—●—).

cols (FairplayMP and VIFF) increase faster (approximately 6–7 times higher in our evaluation interval). Moreover, VIFF results in a timing exception for 200+ participants.

6. Related Work

The authors of Fairplay [9] analyse their framework in local and wide area network settings using functions for logical AND, the billionaire problem, a median function, and a keyed database search function. Henecka *et al.* [15] (in considering Tasty) present statistical information regarding setup and online time in the benchmarking of use cases for set intersections and face recognition, and compare their performance results on multiplication circuits to Fairplay. In the multi-party setting, Ben-David *et al.* [16] (in considering FairplayMP) present benchmarking results deploying a second price auction and a voting scenario. The authors of VIFF [17] include benchmarking results on the secret sharing of random bits.

Küçük *et al.* [23] explore the use of SGX for secure multi-party applications with a focus on the smart grid. In their evaluation, they benchmark basic SGX operations and show that a single SGX-based TRE may support approximately 20,000 relying parties in a 30 minutes interval. Recently, Bahmani *et al.* [24] showed how to utilise isolated execution environments for MPC and introduce a new SGX-based protocol. Furthermore, they give a comparative evaluation using the ABY framework [25]. Nevertheless, to the best of our knowledge, other than [24], there has not been a thorough analysis of state-of-the-art MPC protocols in comparison to SGX-based MPC.

7. Conclusion

We set out to compare the Trustworthy Remote Entity (TRE) concept to state-of-the-art two-party and multi-party protocols. The TRE utilises isolated execution environments to deploy secure multi-party execution. We have shown

TABLE 4: Results for the benchmarking of the multi-party protocols for the operation sum^5 . Results are stated in seconds.

Name/Participants	10	100	200	300	500	700	1000
PlainMP	0.015	0.12	0.24	0.35	0.59	0.82	1.19
TREMP	0.142	1.09	2.06	3.10	5.16	7.23	10.40
FairplayMP	2.32	6.84	12.55	18.77	33.01	50.24	73.42
VIFF	3.66	19.83	-*	-	-	-	-

* Exceeds timing limitation by our benchmarking framework.

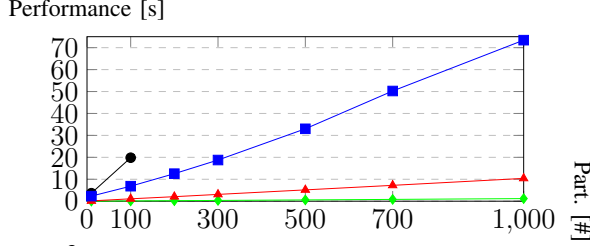


Figure 3: Comparison in the multi-party setting for the operation sum . Included multi-party protocols: PlainMP(◆), TREMP(▲), FairplayMP(■) and VIFF(●).

benchmarking results based on privacy-preserving data mining operations and demonstrated that multi-party protocols are in general independent, in the sense of the execution time, from their functionality. In other words, the performed functionality does not significantly influence the overall execution time. Our benchmarking was performed in two settings: in the two-party setting, we increased the number of operations; in the multi-party setting, we increased the number of participants. The performance evaluation showed that a SGX-based TRE has similar performance to a plain implementation in the two-party setting, and outperforms general multi-party protocols by a factor of approximately 6 in our measure interval (up to 1,000 participants).

We have considered only a centralised TRE. In the future, we will evaluate the extent to which a heterogeneous- or homogeneous-split TRE may further reduce the computational and communication load of secure multi-party protocols.

Acknowledgments

The first author is supported by a research grant from the Intel Corporation.

References

- [1] A. Paverd, “Enhancing Communication Privacy Using Trustworthy Remote Entities,” D.Phil dissertation, University of Oxford, 2016.
- [2] R. Ankele, A. Küçük, A. Martin, A. Simpson, and A. Paverd, “Applying the Trustworthy Remote Entity to Privacy-Preserving Multiparty Computation: Requirements and Criteria for Large-Scale Applications,” ser. ATC 2016. IEEE, 2016.
- [3] Y. Lindell and B. Pinkas, “Secure Multiparty Computation for Privacy-Preserving Data Mining,” *Journ. of Privacy and Confidentiality*, vol. 1, no. 1, pp. 59–98, 2009.
- [4] A. Yao, “Protocols for Secure Computations,” ser. SFCS 1982. IEEE Computer Society, 1982, pp. 160–164.
- [5] O. Goldreich, S. Micali, and A. Wigderson, “How to Play ANY Mental Game,” ser. STOC 1987. ACM, 1987, pp. 218–229.
- [6] D. Beaver, S. Micali, and P. Rogaway, “The Round Complexity of Secure Protocols,” ser. STOC 1990. ACM, 1990, pp. 503–513.
- [7] M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computation,” ser. STOC 1988. ACM, 1988, pp. 1–10.
- [8] M. Rabin, “How to Exchange Secrets by Oblivious Transfer,” Aiken Computation Laboratory, Harvard University, Tech. Rep. 81, 1981.
- [9] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, “Fairplay — A Secure Two-Party Computation System,” ser. SSYM 2004. USENIX Association, 2004, pp. 20–20.
- [10] A. Shamir, “How to Share a Secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
- [11] U. Maurer, “Secure Multi-party Computation Made Simple,” ser. SNC 2002, S. Cimato, G. Persiano, and C. Galdi, Eds., vol. 2576. Springer, 2003, pp. 14–28.
- [12] R. Cramer, I. Damgård, and J. Nielsen, “Multiparty Computation from Threshold Homomorphic Encryption,” ser. EUROCRYPT 2001, B. Pfitzmann, Ed., vol. 2045. Springer, 2001, pp. 280–300.
- [13] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty Computation from Somewhat Homomorphic Encryption,” ser. CRYPTO 2012, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, 2012, pp. 643–662.
- [14] R. Cramer, I. Damgård, and U. Maurer, “General Secure Multiparty Computation from any Linear Secret-Sharing Scheme,” ser. EUROCRYPT 2000, B. Preneel, Ed., vol. 1807. Springer, 2000, pp. 316–334.
- [15] W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg, “TASTY: Tool for Automating Secure Two-party Computations,” ser. CCS 2010. ACM, 2010, pp. 451–462.
- [16] A. Ben-David, N. Nisan, and B. Pinkas, “FairplayMP: A System for Secure Multi-party Computation,” ser. CCS 2008. ACM, 2008, pp. 257–266.
- [17] I. Damgård, M. Geisler, M. Krøigaard, and J. Nielsen, “Asynchronous Multiparty Computation: Theory and Implementation,” ser. PKC 2009, S. Jarecki and G. Tsudik, Eds., vol. 5443. Springer, 2009, pp. 160–179.
- [18] Y. Huang, D. Evans, J. Katz, and L. Malka, “Faster Secure Two-party Computation Using Garbled Circuits,” ser. SEC 2011. USENIX Association, 2011, pp. 35–35.
- [19] D. Bogdanov, S. Laur, and J. Willemson, “Sharemind: A Framework for Fast Privacy-Preserving Computations,” ser. ESORICS 2008, S. Jajodia and J. Lopez, Eds., vol. 5283. Springer, 2008, pp. 192–206.
- [20] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, “SEPIA: Privacy-preserving Aggregation of Multi-domain Network Events and Statistics,” ser. SEC 2010. USENIX Association, 2010, pp. 15–15.
- [21] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell, “SCAPI: The Secure Computation Application Programming Interface,” Cryptology ePrint Archive, Report 2012/629, 2012.
- [22] Y. Lindell and B. Pinkas, “Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer,” ser. TCC 2011, Y. Ishai, Ed., vol. 6597. Springer, 2011, pp. 329–346.
- [23] A. Küçük, A. Paverd, A. Martin, N. Asokan, A. Simpson, and R. Ankele, “Exploring the Use of Intel SGX for Secure Many-Party Applications,” ser. SysTEX 2016. ACM, 2016, pp. 5:1–5:6.
- [24] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A. Sadeghi, G. Scerri, and B. Warinschi, “Secure Multiparty Computation from SGX,” Cryptology ePrint Archive, Report 2016/1057, 2016.
- [25] D. Demmler, T. Schneider, and M. Zohner, “ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation,” ser. NDSS 2015. The Internet Society, 2015.

Appendix A.

Two-Party and Multi-Party Protocol Software

TABLE 5: Software versions and references of the two-party and multi-party protocol software used in our benchmarking.

Name	Type	Version
Fairplay	two-party	v1.0**
Tasty	two-party	v0.1.4 [†]
FairplayMP*	multi-party	v1.1 [‡]
VIFF	multi-party	tip*

* FairplayMP was rewritten by the authors of the paper to be executable in a local setting.
 ** http://cs.huji.ac.il/project/Fairplay/Fairplay/Fairplay_Project.tar
[†] <https://github.com/tastyproject/tasty>
[‡] <https://github.com/FairplayMP/FairplayMP>
 * <http://hg.viff.dk/viff/archive/tip.zip>

Appendix B.

Implementation Strategies in Pseudo Code

TABLE 6: Implementation strategies illustrated in pseudo-code for our different evaluation cases: plain, TRE and MPC.

Plain	TRE	MPC
$A, B:$ $A \rightarrow B:$ send x_a $B:$ eval. $y = f(x_a, x_b)$ $B \rightarrow A:$ return y	$A, TRE:$ $A \rightarrow TRE:$ initiate RA $TRE \rightarrow A:$ respond RA $A \rightarrow TRE^*:$ send x_a $TRE:$ eval. $y = f(x_a, x_b)$ $TRE \rightarrow A:$ return y	$A, B:$ $A, B:$ est. GC; gen. sec. shares $A \rightarrow B:$ send GC; sec. shares $B:$ eval. GC; sec. shares $B \rightarrow A:$ return y
$p_i \quad :$ $i=0..m$ $p_i \rightarrow p_m:$ send x_i $i=0..m-1$ $p_m:$ eval. $y = f(x_i)$ $p_m \rightarrow p_i \quad :$ $i=0..m-1$ return y	$p_i \quad :$ $i=0..m$ $p_i \rightarrow TRE:$ initiate RA $i=0..m-1$ $TRE \rightarrow p_i \quad :$ respond RA $i=0..m-1$ $p_i \rightarrow TRE^*:$ send x_i $i=0..m-1$ $TRE:$ eval. $y = f(x_i)$ $TRE \rightarrow p_i \quad :$ return y $i=0..m-1$	$p_i \quad :$ $i=0..m$ $p_i \quad :$ est. GC; gen. sec. shares $i=0..m$ $p_i \rightarrow p_j \quad :$ send GC; sec. shares $i=0..m \quad j=0..m/i$ $p_i \quad :$ eval. GC; sec. shares $i=0..m$

* Assuming that A trust the TRE . Same applies in the m-party setting.

2-party: In the plain case, we assume for either, the 2-party and m-party setting, that one party (B, p_m) acts as a trustworthy server. This entity computes the function, and returns the result to all other parties.

m-party: The TRE represents a party and acts as an intermediary to all other parties. In the multi-party setting, dependent on the underlying protocol, either garbled circuits (GC) or secret shares are generated, shared and evaluated.