2017

Multiparty computations in varying contexts

Paul Laird

Sarah Jane Delany

Pierpaolo Dondio

Multiparty Computations in Varying Contexts

Paul Laird
Dublin Institute of Technology
Email: paul.laird@dit.ie

Sarah Jane Delany
Dublin Institute of Technology
Email: sarahjane.delany@dit.ie

Pierpaolo Dondio
Dublin Institute of Technology
Email: pierpaolo.dondio@dit.ie

***Abstract*—Recent developments in the automatic transformation of protocols into Secure Multiparty Computation (SMC) interactions, and the selection of appropriate schemes for their implementation have improved usabililty of SMC. Poor performance along with data leakage or errors caused by coding mistakes and complexity had hindered SMC usability. Previous practice involved integrating the SMC code into the application being designed, and this tight integration meant the code was not reusable without modification. The progress that has been made to date towards the selection of different schemes focuses solely on the two-party paradigm in a static set-up, and does not consider changing contexts. Contexts, for secure multiparty computation, include the number of participants, link latency, trust and security requirements such as broadcast, dishonest majority etc. Variable Interpretation is a concept we propose whereby specific domain constructs, such as multiparty computation descriptions, are explicitly removed from the application code and expressed in SMC domain representation. This mirrors current practice in presenting a language or API to hide SMC complexity, but extends it by allowing the interpretation of the SMC to be adapted to the context. It also decouples SMC from human co-ordination by introducing a rule-based dynamic negotiation of protocols. Experiments were carried out to validate the method, running a multiparty computation on a variable interpreter for SMC using different protocols in different contexts.**

I. Introduction

Secure multiparty computation (SMC) is a growing domain of both pure and applied research interest. It allows multiple parties to learn the result of some computation on data held by the parties, without any party revealing their data. The only information which may be gained by a dishonest participant is that which may be deduced from the function output and input known to the participant. There are several different general-purpose SMC schemes, both for binary and arithmetic computations, and families of more specialised protocols for calculating the result of a particular function on the inputs of the parties. While specialised protocols are less flexible, they can be faster than general purpose SMC, depending on context. Some specialised protocols also offer advantages in terms of security thresholds. Different contexts impact various protocols to differing degrees. Some protocols can scale easily in the number of participants, while others cannot, but scale well in the message space (the range of values acceptable as input). Some are computationally bound, while others suffer severely from network latency issues. Some provide higher thresholds of tolerance for dishonest parties while others provide greater security guarantees for a given threshold. There is no one protocol which is most suitable in all contexts.

978-1-5090-5569-2/17/$31.00

It is frequently the case that the SMC logic is baked into application code, and the choice of protocol is based on what was suitable for the first use of the application, limiting reusability of programs. More recently, the SMC logic is represented in an abstract language at design time, but with either a predetermined implementation strategy [12], or an implementation baked into the configuration at deployment time [37]. We argue that for some applications, a context-neutral representation of secure multiparty computations is more appropriate, allowing the detailed interpretation of the computation to be selected based on the relevant contexts. Our contribution is a framework in development, in which multiparty computations are requested on behalf of client applications. Protocols are dynamically and autonomously negotiated between the various parties using the framework, and orchestrated using the most suitable protocol for the context which provides the necessary security guarantees.

In section II we examine state of the art in SMC frameworks, section III sets out our goals and contribution, section IV describes our feasibility analysis and experiments and section V lists conclusions and future work.

II. Background and Related Work

The first realisation of Secure Multiparty Computation was introduced by Yao with the millionaires' problem [43], evaluating an inequality over a limited input space with computational security, using a one-out-of-n oblivious transfer. Since then, many variations have emerged, and advances have been realised, in efficiency, security and adversary model. Schemes have been based on not just garbled circuits, but also secret shares, including boolean [17], additive arithmetic [4], [14] or Shamir [40]. Specialised adaptations include anonymity [42], deniability [6] and covert operation [7].

A. Classification of Secure Multiparty Computation Protocols

While many different forms of SMC exist, they can be broadly categorised as being general purpose or specialised. General purpose protocols represent arbitrary non-branching computations as arithmetic circuits, typically using Beaver's efficient multiplication [1]; boolean circuits using the GMW [17] protocol, or binary circuits garbled using derivatives of Yao's [43] original technique. Many specialised protocols for arithmetic or statistical operations trade generality for greater efficiency, increased collusion tolerance or other advantages.

B. Multiparty Computation in different contexts

Existing practice, until recently, in multiparty computation involved one of: Trusted computation, Universal application of SMC, or custom coding. Mixed-mode programming and flexible implementation are two recent trends making secure multiparty computation more practical and usable. Delegating computation functions to a trusted party is not SMC, but is appropriate in some circumstances.

1) Universal SMC: Domain specific languages for some time [21], [31], [34] sought to capture intentions of SMC programmers at a higher level. Computations are described in a language for secure multiparty computation, and all computation is performed collaboratively in a privacy-preserving manner. This guarantees that all computations are privacy-preserving, but may be inefficient [20].

2) Custom coded approach: A program involving multiparty computations, one or more of which must be carried out in a privacy-preserving manner, is coded such that those computations requiring secrecy are carried out using some SMC protocol which fulfills the requirements. The protocols may be hand-coded into the application, or generated with tool assistance [20], reducing errors, but requiring expertise on the part of the programmer, such as binary circuit description.

3) Mixed-mode programming: Wisteria [38] improves the tradeoff between ease of programming and efficiency where not all operations need to be carried out securely, allowing very similar lines of code, labelled parallel (local, non-SMC) and secure, for programs which carry out the same instructions in a secure multiparty or local open fashion.

4) Flexible implementation: A description of the computation to be performed is provided in terms of an SMC API [37], and is closest to our work. The SMC API includes operations which can be performed by garbled circuits, boolean and arithmetic secret sharing: `addition, multiplication`; boolean sharing and garbled circuits: `subtract, and, xor, mux, equality, greater-equal`. The ability to select protocols on a per-node basis is facilitated by efficient conversions between arithmetic, boolean and Yao shares [12]. Previous work treating the two-party paradigm established that, even for just decisions involving only two schemes, the computational and network costs would be difficult to optimise by hand [23].

Computational, network, or financial (e.g. computation + network costs on Amazon) costs are used for two-party computation protocol selection [37]. Selection by number of participants, collusion threshold, message space size, repeatability, auditability or security model, is not seen in the literature.

C. Comparison of frameworks for SMC

There are a range of frameworks to support secure multiparty computation available, with differing characteristics, strengths, weaknesses, and appropriate circumstances of use.

Table I lists some important features of frameworks for secure computation. Some can accommodate an unbounded (except computationally) number of participants, while others are restricted to two or three-party computation. Almost all of the frameworks listed here use an associated language which abstracts from the complexity of multiparty computation.

1) Types of Secure Computation: Several of the frameworks offer computation on integers distributed as uniformly random additive shares in $\mathbb{Z}_p$ or $\mathbb{Z}_{2^\ell}$; these require secure channels for privacy in the $n > 2$ models. Additive shares do not tolerate misbehaving parties, and can be thought of as an n-out-of-n sharing, although not an issue where $n = 2$. Replication using pseudorandom secret sharing [9] allows larger (*approx.* $3 < n < 15$) groups tolerating a small number of misbehaving parties. Shamir secret sharing [40] also requires secure channels for privacy, and can tolerate $t < \frac{n}{3}$ corrupted parties, the maximum for the information-theoretic setting. Boolean shares are single bits, shared in a similar manner so that any $k > t$ parties can reconstruct each bit. Yao garbled circuits and values are encrypted by one party, with keys for the other party's values obliviously transferred to that party. Homomorphic encryption allows blind computations on encrypted data supplied by another party. Schemes for future inclusion in our framework are marked 'f' in table I.

2) Specialised Protocols: Specialised protocols for many functions outperform generic formulations of the equivalent function in general-purpose secure multiparty computation schemes. Only ObliVM [28] offers explicit support for specialised protocols, though its authors caution against their use as a waste of developer and cryptographer time. If however, concerns surrounding privacy and profiling of online footprints [24], [33] and DNA [13] continue to grow, then mitigation measures [11], [35] will increase in importance. Many specialist protocols, such as privacy-preserving classifiers [5], perform far better than standard privacy-preserving implementations of the basic algorithms. The development of high performance specialised protocols for secure multiparty computation of statistical functions is seen as a priority in a European Commission sponsored SMC publication [41].

3) Mixed Protocols: Some frameworks allow interactions to be built as trees of nodes in which different SMC protocols are used. The ability to convert shares from one form to another [9] facilitates a much wider selection of possible solutions, where one part of a computation uses one form of shares, which are converted to another form for subsequent computation. Performance evaluations of the ABY [12] framework showed these often outperform single protocol solutions.

4) Automatic Selection: While ABY supports these mixes of SMC schemes, it is the responsibility of the programmer to specify which to use for what portions of the computation, a task which grows considerably with circuit size, particularly difficult if the programmer is not a domain or SMC expert. CheapSMC [37] uses heuristics to assign an efficient combination of SMC schemes to the nodes of the computation.

5) Parties, Security model, Thresholds: Many frameworks are restricted to two parties, and each must assume the other is corrupt. For frameworks supporting three or more parties, however, the number and type of corruption which may be tolerated is important, as it may determine whether a computation may proceed, for a particular set of parties. Frameworks

TABLE I: Feature Comparison of Secure Multiparty Computation Frameworks

Framework	Supported Parties	Abstract Definition	Additive	Shamir	Boolean	Yao	Homomorphic	Mixed Protocols	Specialised Protocols	Automatic Selection	Threshold/Security	Secure Channels	Customisable Thresholds	Oblivious RAM	Dynamic Negotiation
Sharemind [4]	3	√	√	×	×	×	×	×	×	×	$1p$	√	×	×	×
VIFF [10], [15]	≥ 2	√	√	√	×	×	√	×	×	×	v	√	√	×	×
FairPlay [2], [31]	n	√	×	×	√	√	×	×	×	×	$\frac{n}{2}p$	×	×	×	×
TASTY [18]	2	√	×	×	×	√	√	×	×	×	p	×	×	×	×
Huang *et al.* [20]	2	×	×	×	×	√	×	×	×	×	p	×	×	×	×
VMCrypt [30]	2	√	×	×	×	√	×	×	×	×	p	×	×	×	×
PICCO [45]	≥ 3	√	√	√	×	×	×	×	×	×	$\frac{n}{2}p$	√	×	×	×
Wisteria [38]	n	√	×	×	√	×	×	×	×	×	$\frac{n}{2}p$	×	×	×	×
ObliVM [27], [28]	2	√	×	×	×	√	×	√	×	×	p	×	×	√	×
Obliv-C [44]	2	√	×	×	×	√	×	×	×	×	p	×	×	×	×
Frigate [32]	2	√	×	×	×	√	×	×	×	×	p	×	×	×	×
PCF [25]	2	√	×	×	×	√	×	×	×	×	v	×	×	×	×
CBMC [19]	2	√	×	×	×	×	√	×	×	×	p	×	×	×	×
ABY [12]	2	√	√	×	√	√	×	×	√	×	a	×	×	×	×
CheapSMC [37]	2	√	√	×	√	√	×	×	√	√	a	×	×	×	×
Our framework	n	√	√	f	f	f	f	√	√	√	v	v	√	×	√

actively secure against malicious or byzantine adversaries are marked 'a' in table I; those which provide passive protection against semi-honest adversaries are denoted 'p'. PCF [25] is adversary-model agnostic, denoted 'v'. Some schemes require authenticated secure channels between all parties, including by definition those providing information-theoretic security. There is no advantage, and frequently a performance penalty in forcing a higher threshold or security level than is considered adequate, motivating variable thresholds, as in VIFF [10].

6) Oblivious RAM: Almost all available SMC frameworks operate on the basis of a circuit, with all inputs being treated equally, meaning searching data requires time $\mathcal{O}(n)$ in the size of the dataset. Oblivious RAM [16] avoids linear overheads, but the large constant terms in asymptotically sublinear solutions remained an obstacle until a two-server model was proposed [29]. ObliVM [28] is the only secure multiparty computation framework which explicitly supports private computations using oblivious RAM.

7) Dynamic Negotiation: Prior to the development of the frameworks discussed, the main effort involved in carrying out a secure multiparty computation consisted of its manual translation into code with no support built in. This has, for the most part, changed, so that once an interaction has been specified in a suitable language, it can be deployed and executed without delay. Research continues to improve the efficiency of the underlying protocols, but the issue of deployment and agreement to perform the computation has yet to be addressed. If secure multiparty computation is used to mitigate the potential effects of an advanced persistent threat, then ad-hoc solutions are viable. For multiple parties across different administrative domains, this is not possible however, and the bottleneck becomes the deployment, authorisation, and coordination of protocol commencement. We are not aware of any other currently available framework supporting the dynamic request for and negotiation of parameters of a desired multiparty computation, necessary for on-the-fly execution of desired multiparty computations.

D. Execution Cost Projections

The principle of parametric estimation of computational costs has been shown to be valid for two-party computation [39], and the model could be extended to multiparty contexts, with adjustment where environmental or privacy constraints or resource limitations impact the running time. As protocol running times should leak no information, performance data could be reported to a community repository, such as github, to serve as reference values for future inferences.

III. Our Contribution

A. Goals

Fundamentally, our goal is to allow secure multiparty computations, which are described in abstract SMC languages to

- Run in a wider range of contexts (broadcast, dishonest majority *etc.*);
- Support a greater number of participants than is permitted by state of the art secure multiparty computation frameworks;
- change the protocol used in response to context change;

while ensuring protocols used satisfy all security requirements.

B. Illustrative Motivating Example

Consider the example of three parties computing simple sums of values for an hourly commodity auction, communicating over a local network. The fact that three independent trading companies have servers on a physical LAN is not unusual, as data centre or rack co-location with stock exchange servers allows leased server space command a significant premium [3]. The exceptionally small latency between the companies' machines means that communicational cost is of limited concern to the parties, due to its limited impact on the cost of the protocol. Three party computations could be written in SecreC [21], a C-like language which is optimised for three parties. This would allow a C programmer with limited SMC expertise write and/or modify the program to reflect adjustments to the rules of the auction.

Consider the impact of a fourth company joining the computation. The new company may not trust the existing three not to collude against it, and therefore schemes designed for three parties are not a viable option. The interaction is rewritten in another scheme. A fifth company joins, this time from a geographically remote location, changing the latency and network costs. The revised scheme is now not necessarily the optimal solution, and the system is revised. As others join and leave the interaction, one joins who specifies that the corruption of all but one party must be tolerated, forcing a further revision and change of protocol, and so on. With a human already unlikely to reliably select a near-optimal configuration [23], the state explosion introduced by the additional parameters described makes this infeasible. Heuristic searches have been shown to find solutions which perform far better than a single SMC scheme in many scenarios [37] [23].

C. Solution

We refer to our solution as Variable Interpretation of Secure Multiparty Computation. A proof-of-concept variable interpreter for SMC is written in python using the twisted framework [26]. Protocols, requirement specifications and protocol adaptation logic modules are pluggable, allowing arbitrary extension to new protocols or adaptation of non-core interpreter behaviour. A participant in the network can issue an ad-hoc request for an evaluation, receive replies from the candidate parties indicating whether they will participate, under what constraints; and proceed with an appropriate protocol.

D. Design and Interaction Model

Each node has an interpreter, which communicates with interpreters on remote nodes. From the node's perspective the interpreter is seen as a black box, taking proposals for execution and returning results or failure. The interpreter also stores the variables on which it may compute in collaboration with other parties, with restrictions on what computations may be performed, and security requirements. Variables' values may be set, along with constraints on the use of those variables, upon variable initialization.

A multiparty computation is described in a Javascript Object Notation (JSON)-based format listing the participants in the computation, their named variables, and an abstract syntax tree-based description of the evaluation to be performed. This format is detailed in section III-F. The computation is passed to an interpreter as a proposed evaluation. That interpreter takes on the role of controller for the evaluation, and sends the proposed evaluation to all prospective participant nodes for approval. The participants reply with their conditions of participation, in terms of requirements for thresholds of colluding participants which may be tolerated, adversary model etc., along with their relative preferences for minimisation of CPU, memory and network costs.

The controller selects a protocol which satisfies the constraints which have been imposed by the participants, in a manner so as to minimise the weighted cost in terms of resources. This selection, identified by a unique ID, is sent to participants, along with the proposal, and the communication pattern for the protocol. When all participants are ready, the controller initiates the protocol. The participants communicate with each other as prescribed in the initiated protocol instance, until the computation outputs a value, returned by the controller.

Participation is always based on the agreement of the parties, who check the characteristics of the selected protocol against their requirements, to ensure it is acceptable. The controller therefore has an incentive to honestly present the most suitable protocol with an appropriate communication pattern. It is important to note that the role of controller is not a privileged or trusted role with respect to the computation. The concept of a controller which is not in a privileged position with respect to the participants is not new, in fact it is taken a step further in the *Secure Computation System* [22] of the U.S. National Institute of Statistical Sciences, in that the co-ordinator of the interaction may be unaware of the function being computed by the participants. Although acting as the controller of inter-party communications, the initiator of the interaction is not necessarily the *co-ordinator* of the MPC. This can be for one of two reasons:

- The selected protocol is a decentralised protocol, without a co-ordinator in the security model. In this case there is no need for central co-ordination, and the imposition of one is unnecessary, and may even impact proofs of security.
- The role of the protocol co-ordinator is computationally intensive, for example involving a brute-force search of the message space for a discrete logarithm. In this case the initiator may delegate co-ordination to a more computationally capable party.

One consequence of the consensual nature of interactions is that the only effective attack admitted by our solution using a conservative policy is time wasting, at reputational cost to the attacker. It is clear that any attack against a protocol a participant is prepared to use remains a valid threat where that protocol is employed. This includes the publication, by the co-ordinator, of a false value at the end of the protocol, if the underlying protocol does not guard against this possibility. The variable interpreter depends on the variety and quality of protocols and the accuracy of their descriptions.

E. Protocol Description

An Instantiation of a Secure Multiparty Computation Protocol is represented in our framework by a tuple: $< P, N, T, M, S, C, R >$ many of whose values are fixed by the nature of the computation to be performed, or deterministically negotiated by the participants.

- P is a Unique Identifier (UID), used only to identify the protocol
- N is the number of distinct participants in a protocol execution, This is inherent in the request for an evaluation, unless the evaluation is FairPlay-style with a small number of servers performing a computation for a larger number of clients, who trust the servers not to collaborate to reveal their data.

T is the number of corrupt participants which must be tolerated without compromising the security of the protocol instance, which is the maximum threshold demanded by any participant.

M is the size of the message space. This is determined by the maximum value which must be supported in the computation, for example in binary polling of 1000 participants this would be 10 (bits) while other computations may require 32 bit, 64 bit or custom length integers.

S is a security parameter for computationally secure protocols, in the same way as ECC and integer factorisation based encryption protocols are compared to the number of bits of an AES system of equivalent security.

C is the number of clients in a server-evaluated protocol. This is relevant for systems like FairPlay where a large group of clients trust a smaller group of servers not to collaborate. Where this is not relevant, $C = N$

R is the number of times the protocol is requested to be run. If protocols amortise a set-up cost across multiple iterations, this is used to adjust the cost of these protocols. It is assumed that $R = 1$ if not otherwise stated.

P can be thought of as the abstract protocol, or a *class* or *family* of protocols, specialised by the other parameters.

P also has certain characteristic binary attributes:

- IT: Whether information-theoretic security is provided by the protocol, in the presence of secure channels.
- Broadcast: Whether the protocol is compatible with insecure channels, i.e. it does not render the protocol insecure if all messages are read by all parties, including the adversary.
- Set-up: Is a special set-up required prior to initiation of the protocol? Some protocols require some secrets to be shared prior to commencement of the protocol, typically yielding a simplified, efficient protocol. The cost of providing the setup is then amortised over several instances of the protocol.
- Self-opening: Can any participant extract the result from the communication record, or is a special collaborative operation required to open the result?
- Owner-Evaluators: Are the data owners the evaluators of the computation?
- Malicious: Can arbitrary behaviour by up to T participants be tolerated without compromising the privacy of participants' data?
- Multi-Run: Is the amortised cost of the protocol diminished when it is run multiple times with the same participants?

There are also further characteristics which can be expressed as a value or a function of protocol parameters, such as:

- Computational Cost: expressed as a function of number of participants N, solution space bits M (Particularly relevant for computations in the exponent, which are then subject to calculation of a discrete log), and Threshold T.
- Communicational Cost: in messages per participant. Relevant in low-power wireless scenarios to minimise radio transmission costs.
- Rounds of Communication: For high-latency connections this is a limiting factor.
- Bandwidth: Communicational Cost in bytes.
- Entropy: Of relevance where the entropy available to the device is limited or known (e.g. embedded sensors).
- Memory: cost per participant - again relevant in resource-limited scenarios.

F. Co-ordination Protocol

Without loss of generality we use the term *participants* in the sense of owner-evaluators. The parties who supply input data execute the SMC protocol. The controller receives an evaluation request and forwards the proposed evaluation to the other participants. The proposal is structured as follows:
{<tag>, <uid>, <participants>, <operands>, <function>}
Where <tag> identifies the type of message; <uid> is a unique identifier assigned to the evaluation; <participants> is a list of participants details, where the <participants> [index] may be used to identify each participant thereafter; <operands> is a list of inputs to the proposed function, identifying which participant owns the data, and a label for the data; <function> is an abstract syntax tree based description of the function to be evaluated. It is outside the scope of this work to determine how labels should be mapped to local variables by the participant, except to note that in the entirely plausible scenario where parties use the same software system, this would be straightforward. Manual mappings to local variables is even preferable to manual intervention to load data at runtime.
The participants respond with:
<tag>, <uid>, <pid>, <response>, <constraints>
Where <pid> identifies the participant; <response> is a boolean indicating willingness to proceed with the computation, subject to all security criteria being met; <constraints> are the constraints which must be satisfied for the computation to take place, for example {"t":4} would indicate that default parameters are acceptable, so long as the threshold of collusion tolerance exceeds four corrupted parties (t is used to denote the threshold for collusion tolerance). Constraints may also indicate the maximum number of CPU cycles required, or some other resource-based metric, for resource-constrained devices, along with (advisory) preferences for the relative weighting of CPU, memory, communication bytes and communication rounds in picking the protocol from the shortlist satisfying all actual constraints.
The controller selects a suitable protocol. If a participant has declined to participate, the controller may issue a revised proposal, taking account of the departure, at the same time as suggesting a protocol. Otherwise, the controller selects a protocol by identifying the subset of applicable protocols which satisfy constraints imposed by the participants, and

selecting the protocol from that set which minimises a weighted cost metric. It then sends a message as follows:
{<tag>, <uid>, <protocolID>, <setup> }
Where <protocolID> is a unique identifier for the protocol, hashed with any applicable version number to ensure no incompatibility; <setup> is the proposed communication pattern and any other setup information required for any protocol which is not deterministic or symmetric in its configuration. <participants>, <operands>, <function> may also be re-sent if adjustment is necessary, and in such case participants may alter their acceptance criteria for the computation.
The participants, upon receiving this message, if they are to take part, i.e. if the protocol is acceptable, then set up the protocol and prepare to handle protocol-specific messages, before responding positively.
The controller then sets up the protocol, prepares to receive the final values, and initiates the protocol execution. If applicable the controller distributes the final value after the computation has finished.

G. Restrictions

Certain types of SMC are by definition excluded from the available protocols. Covert multiparty computations [7] must be arranged in advance and hide participation unless the result is favourable - seeking the computation leaks information in this scenario, so it cannot be used with the variable interpreter. Protocols incompatible with the operation of the variable interpreter, were not considered further.

IV. Evaluation

The goal of the experiments is neither to demonstrate nor compare the performance of the protocols, which can be reasonably inferred from theoretical calculations, but rather to demonstrate the feasibility of dynamically negotiating the protocol, and the utility of changing to a better protocol where a change in context justifies it. An appropriately selected protocol is more desirable than an arbitrarily chosen or statically selected protocol across all contexts. In this initial work a limited number of protocols are utilised, but the principle is to demonstrate feasibility before expansion to a wider set of protocols.

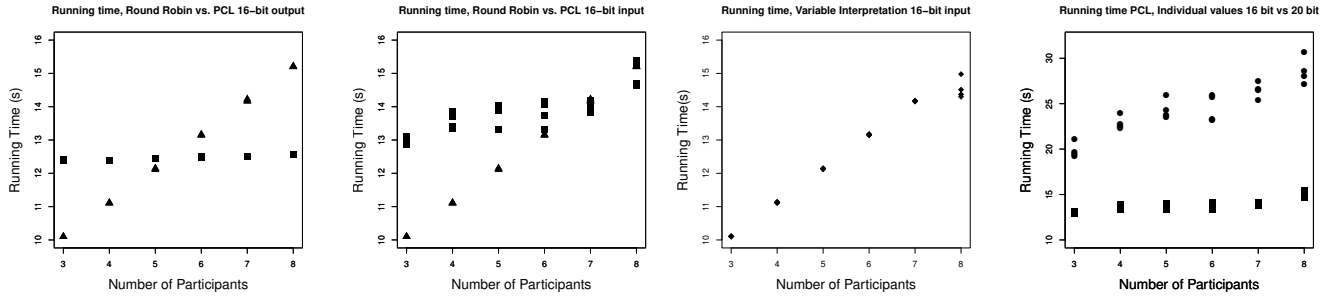
Executions of secure multiparty computations using the framework were carried out on a DELL Latitude E5440 with 4GB RAM and a Dual Core Intel® Core™ i3-4030U CPU @ 1.90GHz running Ubuntu 16.4. Each replicate involved a protocol being executed ten times, and the time for ten evaluations being recorded, and four replicates were run for each set-up. Experiments were run in the context of a simulated wide area network, with latencies simulated by introducing a delay of 100ms on delivery of packets. This is, to a large extent, responsible for the relatively long execution times, but this is a realistic level of delay for executions across continents.

Protocol 1 was an Information-Theoretic (requiring point-to-point secure channels) secure, computationally light protocol requiring a number of rounds of communication linear in the number of users, which offers no protection against colluding adversaries [8], denoted Round Robin. Protocol 2 was a computationally secure, computationally expensive protocol requiring two rounds of protocol communication per computation, secure against $n-2$ colluding adversaries [36] denoted PCL. The framework was initially set up to force selection of protocol 1 or protocol 2 for the experiments, followed by executions where the protocol was selected based on the parameters.

The results of specific protocol executions are graphed in figure 1, with protocol 1 denoted by triangles, and protocol 2 denoted by squares in figures 1a and 1b. One element of the context, the number of participants, had a significant impact on which protocol performed better. In the case where output message space was fixed at 16 bits (Figure 1a), with up to five participants, protocol 1 was more efficient than protocol 2, but with six or more participants, protocol 2 was more efficient. Another element of the context, message space, impacted performance of protocol 2 but not protocol 1. Where input message space was fixed, output message space is n times larger, and protocol 1 remained more efficient below 7 participants (Figure 1b). Where protocol 2 was executed with input message space of 20 bits, its execution times increased greatly, as shown by the circles, in contrast to the squares (16 bit input) in Figure 1d

Average running time for 10 evaluations of an integer sum in the variable interpreter, with 100ms link latency (to simulate Wide-Area-Network performance), n participants, were as follows:

Protocol 1 execution times varied to a large degree with changing number of participants, and had a minimum execution time for the ten evaluations of 10.1 for $n=3$, and maximum of 15.22 for $n=8$, unaffected by bit length of the message space. Protocol 2 had execution time which varied mostly with the bit length of the result. For a fixed maximum result in 16 bits, it showed little variation in execution time with changing n, with minimum execution time of $12.37s$ and maximum of $12.58s$. Protocol 2 execution times varied considerably more with bitlength of the message space. When input bitlength is fixed, output bitlength grows with $log_2(n)$. For a maximum input bitlength of 16, i.e. maximum resulting value of $n \times 2^{16}$, it had minimum execution time of $12.87s$ for $n=3$ and maximum of 15.4s for $n=8$. For a maximum input bitlength of 20, minimum execution time was 19.2s for $n=3$ and maximum execution time was $30.6s$ for $n=8$. The execution involving protocol selection was run with parameters derived from the results for individual protocols and simplified: Communicational cost: Protocol 1: $latency*(n+1)$, Protocol 2: $4*latency$; Computational cost: Protocol 1: 0, Protocol 2: $100+\frac{\sqrt{m}}{2}$. With equal weighting of communication and computation, this resulted in the most efficient protocol being executed except possibly for the point at which the performance is approximately equal: $n=7,\ m=n*2^{16}$. The performance under variable interpretation was equal, within the variance exhibited, to the relevant single protocol experiment for that number of participants. Minimum



(a) RR vs PCL 16-bit output (b) RR vs PCL 16 bit input (c) Variable Int. 16 bit input (d) PCL 16 vs 20 bit input

Fig. 1: Time to negotiate and execute $\sum_1^n v_i$ 10 times, where each value v_i is held privately by participant p_i in various contexts

execution time was $10.11s$ for $n = 3$ and maximum was $14.98s$ for $n = 8$ (Figure 1c).

It is not necessary to evaluate these protocols for different collusion thresholds, as protocol 1 has no tolerance for inter-party collusion. If collusion must be tolerated, protocol 2 must be used rather than protocol 1. On the other hand if numbers of large bitlength were also required, the discrete logarithm would become intractable, due to its $\sqrt{m}$ (where message space $m = 2^{bitlength}$) theoretical complexity. In such circumstances a different protocol would be required.

A. Relationship to Theoretical Values

The experiments were carried out using the pessimistic 6 round negotiation, so for protocol 1, as expected, the time taken is dominated by the $6 + (n + 1)$ times network latency, reflecting the required rounds of communication. For protocol 2, there is a significant computation in the final discrete logarithm, and while there is no n term in the number of rounds of communication, the baseline is 5. The performance of this protocol was in line with expectations, in that time taken, over 11 times latency, rose with message space m, at approximately $\mathcal{O}(\sqrt{m})$. While the crossing point in figure 1a indicates an advantage for protocol 2 for high numbers of participants, the advantage would be reduced at lower link latency. The time cost for protocol 1 also does not increase with the size of the solution space, unlike protocol 2, where the discrete logarithm dominates the execution time for that protocol for large numbers - in figures 1b and 1d the solution space grows with n. The effect of this can also clearly be seen in a comparison between 16 bit (square) and 20 bit (round) input values (figure 1d)

B. Limitations of the Results

The results were obtained by executing several independent processes on a single machine, and non-optimal implementations of protocols were used. Protocols incur an overhead from the execution of the protocol negotiation, which is currently set to prefer extra rounds of negotiation to incurring spurious setup costs. A variation of the co-ordination protocol, employing optimistic negotiations, where the controller makes assumptions about the likely requirements of the participants, and participants which do not reject the proposal assume all parties accept it, would save 4 rounds of communication. This would be more appropriate in the high latency context, while avoiding spurious set-up costs may justify the longer negotiation in low-latency environments. The degree of variation in evaluation times suggests that a sufficiently volatile context justifies the overhead. The two protocols themselves are chosen for illustrative purposes, being opposites in many characteristics.

V. Conclusions and Future Work

The feasibility of variable interpretation of simple secure multiparty computation programs, including dynamic negotiation of the protocols used has been demonstrated, with two different protocols being automatically executed to carry out the same command in different contexts. A marked difference in relative performance is noted between the two protocols for different parameters, with the relative advantage determined in this case by the number of participants in the computation, and solution space. Much work remains to be completed to fully demonstrate the practical utility of the virtual interpreter with respect to SMC, but its feasibility has been demonstrated in principle. Existing work for the two-party context is well developed in respect of minimisation of computational time, bandwidth, or a monetary function of these, as a metric for the selection of protocols or sharing schemes in a static context. This work motivates and examines the inclusion of thus far unconsidered options such as specialised protocols, and unexplored parameters in the context of protocol selection, such as number of participants, message space, and collusion tolerance threshold; in the dynamic negotiated selection of multiparty protocols for more than two parties.

The integration of further protocols including general purpose schemes is the focus of future work. Integration of general purpose schemes will allow the direct comparison of execution time including negotiation with predetermined execution patterns. Other important future work includes the development of policies and agents to move away from manual authorisation and triggering of all multiparty computations. This is where we envisage the real-world bottleneck will lie, once the technology for protocol execution has plateaued, much as the human decision to trade stock was the slowest

part of a trade following computerisation, leading to the rise in algorithmic automated trading.

REFERENCES

[1] BEAVER, D. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference* (1991), Springer, pp. 420–432.

[2] BEN-DAVID, A., NISAN, N., AND PINKAS, B. FairplayMP: a System for Secure Multi-Party Computation. In *Proceedings of the 15th ACM conference on Computer and communications security* (2008), pp. 257–266.

[3] BOEHMER, E., FONG, K. Y., AND WU, J. J. International evidence on algorithmic trading. In *AFA 2013 San Diego Meetings Paper* (2014).

[4] BOGDANOV, D., LAUR, S., AND WILLEMSON, J. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security* (2008), Springer, pp. 192–206.

[5] BOST, R., POPA, R. A., TU, S., AND GOLDWASSER, S. Machine learning classification over encrypted data. In *NDSS* (2015).

[6] CANETTI, R., AND GENNARO, R. Incoercible multiparty computation. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on* (1996), IEEE, pp. 504–513.

[7] CHANDRAN, N., GOYAL, V., OSTROVSKY, R., AND SAHAI, A. Covert multi-party computation. In *Foundations of Computer Science, 2007. 48th Annual IEEE Symposium on* (2007), IEEE, pp. 238–248.

[8] CLIFTON, C., KANTARCIOGLU, M., VAIDYA, J., LIN, X., AND ZHU, M. Y. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations Newsletter 4*, 2 (2002), 28–34.

[9] CRAMER, R., DAMGÅRD, I., AND ISHAI, Y. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography Conference* (2005), Springer, pp. 342–362.

[10] DAMGÅRD, I., GEISLER, M., KRØIGAARD, M., AND NIELSEN, J. B. Asynchronous multiparty computation: Theory and implementation. In *International Workshop on Public Key Cryptography* (2009), Springer, pp. 160–179.

[11] DE CRISTOFARO, E., FABER, S., GASTI, P., AND TSUDIK, G. Genodroid: are privacy-preserving genomic tests ready for prime time? In *Proceedings of the 2012 ACM workshop on Privacy in the electronic society* (2012), ACM, pp. 97–108.

[12] DEMMLER, D., SCHNEIDER, T., AND ZOHNER, M. Aby–a framework for efficient mixed-protocol secure two-party computation. In *Proceedings of the Network and Distributed System Security Symposium* (2015).

[13] ERLICH, Y., AND NARAYANAN, A. Routes for breaching and protecting genetic privacy. *Nature Reviews Genetics 15*, 6 (2014), 409–421.

[14] GEISLER, M. Viff: Virtual ideal functionality framework. *Homepage: http://viff. dk* (2007).

[15] GEISLER, M. J. B. *Cryptographic Protocols:: Theory and Implementation*. PhD thesis, Aarhus Universitet, 2010.

[16] GOLDREICH, O. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing* (1987), pp. 182–194.

[17] GOLDREICH, O., MICALI, S., AND WIGDERSON, A. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing* (1987), ACM, pp. 218–229.

[18] HENECKA, W., KÖGL, S., SADEGHI, A.-R., SCHNEIDER, T., AND WEHRENBERG, I. TASTY: Tool for Automating Secure Two-partY computations. In *17th ACM Conference on Computer and Communications Security* (2010), pp. 451–462.

[19] HOLZER, A., FRANZ, M., KATZENBEISSER, S., AND VEITH, H. Secure two-party computations in ansi c. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 772–783.

[20] HUANG, Y., EVANS, D., KATZ, J., AND MALKA, L. Faster Secure Two-Party Computation using Garbled Circuits. In *USENIX Security Symposium* (2011), pp. 1–16.

[21] JAGOMÄGIS, R. Secrec: a privacy-aware programming language with applications in data mining. *Master's thesis, University of Tartu* (2010).

[22] KARR, A. F., FULP, W. J., VERA, F., YOUNG, S. S., LIN, X., AND REITER, J. P. Secure, Privacy-Preserving Analysis of Distributed Databases. *Technometrics 49*, 3 (2007), 335–345.

[23] KERSCHBAUM, F., SCHNEIDER, T., AND SCHRÖPFER, A. Automatic protocol selection in secure two-party computations. In *International Conference on Applied Cryptography and Network Security* (2014), Springer, pp. 566–584.

[24] KOSINSKI, M., STILLWELL, D., AND GRAEPEL, T. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences 110*, 15 (2013), 5802–5805.

[25] KREUTER, B., SHELAT, A., MOOD, B., AND BUTLER, K. R. Pcf: A portable circuit format for scalable two-party secure computation. In *Usenix Security* (2013), vol. 13, pp. 321–336.

[26] LEFKOWITZ, G. Twisted Matrix Labs.

[27] LIU, C., HUANG, Y., SHI, E., KATZ, J., AND HICKS, M. Automating efficient ram-model secure computation. In *Security and Privacy (SP), 2014 IEEE Symposium on* (2014), IEEE, pp. 623–638.

[28] LIU, C., WANG, X. S., NAYAK, K., HUANG, Y., AND SHI, E. Oblivm: A programming framework for secure computation. In *Security and Privacy (SP), 2015 IEEE Symposium on* (2015), IEEE, pp. 359–376.

[29] LU, S., AND OSTROVSKY, R. Distributed oblivious RAM for secure two-party computation. In *Theory of Cryptography*. Springer, 2013, pp. 377–396.

[30] MALKA, L. Vmcrypt: modular software architecture for scalable secure computation. In *Proceedings of the 18th ACM conference on Computer and communications security* (2011), ACM, pp. 715–724.

[31] MALKHI, D., NISAN, N., PINKAS, B., AND SELLA, Y. Fairplay—a Secure Two-Party Computation System. In *USENIX Security Symposium* (2004), pp. 287–302.

[32] MOOD, B., GUPTA, D., CARTER, H., BUTLER, K., AND TRAYNOR, P. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on* (2016), IEEE, pp. 112–127.

[33] NARAYANAN, A., AND SHMATIKOV, V. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on* (2008), IEEE, pp. 111–125.

[34] NIELSEN, J. D., AND SCHWARTZBACH, M. I. A Domain-Specific Programming Language for Secure Multiparty Computation. In *Proceedings of the workshop on Programming languages and analysis for security* (2007), ACM, pp. 21–30.

[35] NIKOLAENKO, V., IOANNIDIS, S., WEINSBERG, U., JOYE, M., TAFT, N., AND BONEH, D. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 801–812.

[36] PATSAKIS, C., CLEAR, M., AND LAIRD, P. Private aggregation with custom collusion tolerance. In *Information Security and Cryptology* (2014), Springer, pp. 72–89.

[37] PATTUK, E., KANTARCIOGLU, M., ULUSOY, H., AND MALIN, B. Cheapsmc: A framework to minimize secure multiparty computation cost in the cloud. In *IFIP Annual Conference on Data and Applications Security and Privacy* (2016), Springer, pp. 285–294.

[38] RASTOGI, A., HAMMER, M. A., AND HICKS, M. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *Security and Privacy (SP), 2014 IEEE Symposium on* (2014), IEEE, pp. 655–670.

[39] SCHROEPFER, A., AND KERSCHBAUM, F. Forecasting run-times of secure two-party computation. In *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on* (2011), IEEE, pp. 181–190.

[40] SHAMIR, A. How to Share a Secret. *Communications of the ACM 22*, 11 (1979), 612–613.

[41] SMART, N. P., ARCHER, D., BOGDANOV, D., BOLDYREVA, S., KAMARA, S., KERSCHBAUM, F., LINDELL, Y., LU, S., NIELSES, J. B., OSTROVSKY, R., PAGTER, J. I., SADEGHI, A.-R., AND WALLER, A. Future directions in computing on encrypted data. Tech. rep., 2015.

[42] STAJANO, F., AND ANDERSON, R. The Cocaine Auction Protocol: On the Power of Anonymous Broadcast. In *Information Hiding*, A. Pfitzmann, Ed., vol. 1768 of *Lecture Notes in Computer Science*. Springer, 2000, pp. 434–447.

[43] YAO, A. C. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science* (1982), pp. 160–164.

[44] ZAHUR, S., AND EVANS, D. Obliv-c: A language for extensible data-oblivious computation. *IACR Cryptology ePrint Archive* (2015), 1153.

[45] ZHANG, Y., STEELE, A., AND BLANTON, M. Picco: a general-purpose compiler for private distributed computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 813–826.