Forecasting Run-Times of Secure Two-Party Computation

Axel Schröpfer
SAP Research
Karlsruhe, Germany
Email: axel.schroepfer@sap.com

Florian Kerschbaum
SAP Research
Karlsruhe, Germany
Email: florian.kerschbaum@sap.com

***Abstract*—Secure computation (SC) are cryptographic protocols that enable multiple parties to perform a joint computation while retaining the privacy of their inputs. It is current practice to evaluate the performance of SC protocols using complexity approximations of computation and communication. Due to the disparate complexity measures and constants this approach fails at reliably predicting the performance.**
We contribute a performance model (PM) for forecasting run-times of secure two-party computations. We show the correctness of our PM by an empirical study on the problem of secure division which is relevant for many real world SCs, e.g., k-means clustering or supply chain optimization. We show that our PM can be used to make an optimal selection of an algorithm and cryptographic protocol combination, as well as to determine the implicit security tradeoffs. The predictions of our PM can be used to design or select more efficient or more secure protocols.

***Index Terms*—Multi-party Computation Performance Model Security**

I. Introduction

Numerous implementations for secure computations (SC) exist [1], [2], [3], [4], [5], [6]. Yet, for many applications – even of moderate size – performance remains critical [7].

There is an abundance of specialized protocols for SCs improving performance. Yet, theoretic developments compare different performance metrics. There is computation complexity for the number of computation steps, communication complexity for the size of all messages and round complexity for the number of communication rounds. For a given environment and a given protocol it is almost always unknown which complexity dominates the performance. As a result it is very difficult to pick the best SC for a given problem.

In this paper we try to give some decision help and present a performance model (PM). Our PM captures an algorithm in abstract form and can then forecast a run-time based on a benchmark. The benchmark is performed for an individual environment and captures its basic computation and communication parameters.

We can cover a wide range of algorithms. The motivation for this study was the problem of secure division in JELS [8] or weighted average in secure k-means clustering [9]. We therefore chose two division algorithms as an example [10], [8]. We cover the SC protocols based on garbled circuits (GC) [11] and homomorphic encryption (HE) [12], [13]. We also cover different computation and network settings.

We verified the accuracy of our model in a large empirical study. Then, we used the model to try to predict the future development of performance of SCs. There are a number of conflicting trends, such as increasing input-size and key-lengths. Also different network settings and CPU architecture have a significant impact. This difficulty in making informed decisions underpins the need for an integrated model, such as our PM capable of dealing with all those situations.

A future application of our model is an optimizing compiler that supports both SC protocols, e.g. [5]. The compiler can first transform the algorithm into our abstract model and then select the best SC protocol options based on the forecasts of our PM.

In summary this paper contributes

- an abstract model for algorithms in SCs
- a PM forecasting the run-time of a SC
- an accuracy assessment of the forecast
- a number of predictions for the evolvement of the performance of different SCs

The remainder of the paper is structured as follows. Section 2 contains related work. In Section 3 we introduce the PM which we evaluate for accuracy in Section 4. Section 5 contains a discussion. Section 6 compares PM vs. asymptotic complexities and Section 7 concludes the paper.

II. Related Work

Our work is related to practical SCs [14], [15], [7], [16], secure division [10], [9], [17], [18], SC implementation [5], [1], [4], [19], [3] and performance of cryptographic protocols [20].

Several SCs have been brought to practice. In [14] the authors present a SC for price bidding. The work presents the first commercial SC, but rather states a general feasibility result. In [15], [7] the authors present an experimental study of a specific protocol for collaborative benchmarking. The work discusses the impact of network conditions on SCs, comparing SC run-times in LAN and WAN environments. The authors show for a sub-class of SCs [7] that network can have minor practical impact on the overall run-time of the protocol. Another demonstration of practical feasible SC is the work in [16], showing that AES can be performed as a SC in the malicious model in practically acceptable time.

Our use-case example of secure division has been investaged in scientific literature before. In [10] the authors present secure division approaches for secure two-party and multi-party computation. They use different arithmetic approaches, like scaling and multiplication by the inverse. The inverse is computed following the Newton Raphson approximation. In [18] the authors propose a secure division founded on a sub-protocol for oblivious polynomial evaluation. In [9] the authors propose a secure division method following an adaption of securely subtracting the denominator from the numerator as many times as possible. In [17] the authors use fixed point data representation and NR approx. to realize secure division.

A number of frameworks and tools for SCs have been proposed including performance measurement. In [1] the authors present FairPlay which was the first compiler implementing SC. FairPlayMP [2] extends FairPlay to the multi-party setting. It is based on the protocol of [21] which is also an extension of [11] to the multi-party setting. In [4] the authors present Sharemind, a framework built for experimenting with privacy-preserving data mining. It implements the SC protocol from [22]. It has been optimized for speed of simple operations, such as vector products. In [3] the authors present VIFF, a framework for SC which represents the basis for the first commercial application of SC [14]. It implements multi-party SC in the information-theoretic model [23].

Next to these implementations of one SC protocol implementations of mixed protocols have emerged. These combine several protocols, such as GC and HE. The authors of [5] present a tool which allows to express a computation whose segments are translated in either GC or HE based sub-protocols. The work evaluates an exemplary SC and shows the performance improvement gained by mixing techniques. In [19] the authors introduce a programming language for cryptographic protocols which also allows expressing mixed technique SC protocols. The authors also evaluate an exemplary SC, showing noticeable improvements in run-time using a mixed approach of GC and HE. Our PM can, e.g., be used to select optimal combinations.

There are also PMs for other cryptographic protocols. The authors of [20] consider private information retrieval (PIR) protocols for scenarios with a single server. They investigate client access patterns in order to determine whether it is faster to transfer the overall database and perform rich local computation or to run a protocol and to perform less computation. Similarly to us, the authors propose an arithmetic approach to predict an optimal choice.

III. Performance Model

The PM captures input and output, algorithm, SC protocol and system and allows for a given parameterization to estimate the run-time of an execution.

A SC is implemented as cryptographic protocol and ultimately executed as two interacting computer programs running on two computers connected by a communication network.

We separate our model and its parameters into four layers. The top-most layer captures parameters about inputs and outputs. The second top-most layer captures the formalized algorithm. The third layer captures parameters of the SC protocol where we implement two sub-models, one expressing a SC based on GC, the other one for SCs based on HE. Finally, the SC is executed as a program on a computer system, making OS calls for transmission which are captured in the bottom layer. Tab. I gives an outline of all layers and contained parameters. Once the parameters are set, the PM computes a forecast of the run-time of the SC from the parameters.

A. Input/Output

The input layer of the PM captures the size and number of inputs and outputs. We restrict the model to handle integer values. Let l denote the number of bits of input and output values. We use two-complement notation and the domain for signed integers is $\left[-2^{l-1}, 2^{l-1}-1\right]$. The number of private inputs of A is denoted α^A and the number of private outputs is denoted β^A. Similarly, α^B and β^B denote the private inputs and outputs of B. Consequently, there are $l(\alpha_A + \alpha_B)$ input bits and $l(\beta_A + \beta_B)$ output bits.

B. Algorithm

The algorithm is written in a machine language-like format. We use a single static assignment notation. The algorithm is expressed as a sequential list O of n operations, $O = \{o_1, \cdots, o_n\}$. An operation is represented by a tuple consisting of left operand, operator and right operand, $o = \left(\vec{l}, \circ, \vec{r}\right)$. Operands are scalars with the exception of the operator $\odot_\epsilon$. We implement the following operators:

- addition $\oplus$
- subtraction $\ominus$
- scalar multiplication $\odot_\epsilon$ with operands being vectors of ϵ elements
- multiplication by a constant $\odot_c$
- division by a constant $\oslash_c$
- left shift $\ll$
- right shift $\gg$
- less-or-equal $\leq$

We emphasize that the operators have been carefully chosen, such that the underlying SC protocols can leverage their individual advantages. For example, GC based SC performs well for operators $\ll$ and $\gg$ since those operators are implemented "for free" just by wiring the circuit. Also the $\leq$ operator can be implemented very efficiently since it requires access to single bit. On the other hand, for GC based SC, arithmetic operators need to be implemented as more complex sub-circuits. HE based SC implements arithmetic operations efficiently, in particular $\oplus$, $\ominus$, $\odot_\epsilon$ and $\odot_c$. However, for operators which need access to single bits, a costly decomposition (e.g., [24]) needs to be performed.

C. Secure Computation

The SC layer contains parameters of the cryptographic protocol and combines all parameters into the forecast of the overall SC. In our PM, we implement two SC protocols. The first one is based on GC and uses the cryptographic protocol of Yao [11]. The second one is based on HE and combines protocols of [10], [13], [25] and additive sharing. Both SC

Parameter	Description
Input/Output	
l	Bit-length of an input or output value
ρ	Party with $\rho \in \{A, B\}$
α^ρ	Number of private inputs of ρ
β^ρ	Number of private outputs of ρ
Algorithm	
$o = \left(\vec{l}, \circ, \vec{r}\right)$	Operation on left-hand and right-hand operand vector
$\circ \in \{\oplus, \ominus, \odot_\epsilon, \odot_c, \oslash_c, \lll, \ggg, \leq\}$	Operator,sub-script c explicitly denotes operation on constant, $\lvert\vec{l}\rvert = \lvert\vec{r}\rvert = 1 + z, z = \begin{cases} \epsilon - 1 & \text{if } \circ = \odot_\epsilon \\ 0 & \text{otherwise} \end{cases}$
$O = \{o_1, \cdots, o_n\}$	List of operations
Secure Computation	
k_{OT}	Key-length of Oblivious Transfer
k_{GC}	Key-length of GC
k_{HE}	Key-length of HE
System	
$t^\rho_{RND}(n)$	Time in [ms] to select n random bits
$t^\rho_{ADD}(n)$	Time in [ms] to add two n bit numbers
$t^\rho_{MUL}(n)$	Time in [ms] to multiply two n bit numbers
$t^\rho_{POW}(n)$	Time in [ms] for one modular exponentiation of two n bit numbers
$t^\rho_{INV}(n)$	Time in [ms] to compute modular inverse in a field of n bits
$t^\rho_{OWH}(n)$	Time in [ms] for hash function for one n bit number
$t^\rho_{ENC}(n)$	Time in [ms] for homomorphic encryption with n bit key
$t^\rho_{DEC}(n)$	Time in [ms] for homomorphic decryption with n bit key
t_{LAT}	Network latency in [ms]
b	Network bandwidth in [Mbit/s]
$r_{t_{LAT},b}(n)$	Transfer rate in [Mbit/s] for n bits (depending on bandwidth b and latency t_{LAT})
mtu	Maximum number of bits of payload per network packet

TABLE I
PARAMETERS OF THE PERFORMANCE MODEL

protocols require a common cryptographic protocol for 1-out of-2 oblivious transfer (OT).

1) OT: For 1-out of-2 OT we use the efficient protocol of [26]. The protocol operates in a field of bit-length k_{OT}. Its operations are multiplication, modular exponentiation, and inversion requiring time $t^\rho_{MUL}(k_{OT})$, $t^\rho_{POW}(k_{OT})$ and $t^\rho_{INV}(k_{OT})$, respectively. In addition, the protocol makes calls to a one-way hash function requiring time $t^\rho_{OWH}(k_{OT})$ per call. The time OT requires can be estimated as

$$\begin{aligned} t_{OT} \quad = \quad & 2t^A_{MUL}(k_{OT}) + 5t^A_{POW}(k_{OT}) + 1t^A_{INV}(k_{OT}) + \\ & 4t^A_{OWH}(k_{OT}) + 1t^B_{MUL}(k_{OT}) + 2t^B_{POW}(k_{OT}) + \\ & 1t^B_{INV}(k_{OT}) + 2t^B_{OWH}(k_{OT}) + 4t_{LAT} \end{aligned}$$

2) GC based SC: For Yao's protocol, the algorithm represented as list of operations O needs to be translated into a binary circuit C. From this binary circuit C we require its size, i.e., number of gates n_g in the circuit. Tab. II gives a translation of algorithm operators into number of gates. The size of the circuit can then be simply computed as the sum of all of its operators. Note that this table is just as an estimation and states an upper bound on the number of gates. Optimizations may very well be applied, on the operators itself as well as on the overall circuit, further reducing n_g.

Yao's protocol consists of five phases. We now present the estimation for each phase.

The first phase consists of garbling the circuit. Each wire of the circuit is assigned a random string of length k for each possible value. Let $n_w = \alpha^A + \alpha^B + n_g$ be the number of wires (i.e., all input wires and all binary gate output wires). Let $t^A_{RND}(\kappa)$ be the time for selecting κ random bits. We set

Operator $\circ$	Number of gates n_g
$\oplus$	$5l - 3$
$\ominus$	$11l - 6$
$\odot_\epsilon$	$\epsilon\left(6l^2 - 8l + 3\right) + (\epsilon - 1)(5l - 3)$
$\oslash_c$	$\left(22l^2 - 11l + 5\right)$
$\lll$	0
$\ggg$	0
$\leq$	$5l - 3$

TABLE II
NUMBER OF GATES n_g OF ALGORITHM OPERATOR $\circ$ IN CORRELATION TO l

$\kappa = k_{GC}$. The time of phase one can be computed as

$$t_1 = 2n_w t^A_{RND}(\kappa)$$

After garbling the wires, the output values of each truth table are encrypted. Let t^A_{OHW} be the time for performing one encryption. Considering binary gates, the total time of phase two can be computed as

$$t_2 = 4n_g t^A_{OWH}(\kappa)$$

Phase three consists of encrypting and transmitting the inputs of the B which is achieved using 1-out of-2 OT. There is one OT for each of B's α^B inputs of l bits. Hence,

$$t_3 = \alpha^B l t_{OT}$$

Afterwards, the encrypted circuit is transmitted from A to B, as well as A's encrypted inputs and the output keys. Let n_t denote the bits for the encrypted circuit, n_i the bits for A's encrypted input bits and n_o the bits for the output keys (reference of size k_{GC} per possible value per wire of B's outputs). Let k_{GC} denote the key-length of an encrypted entry in the truth table. The total number of bits to transfer is $n_i + n_t + n_o$ for

$$n_i = \alpha^A l \kappa$$

$$n_t = 4n_g k_{GC}$$

and

$$n_o = 2\beta^B l k_{GC}$$

We use a look-up table which keeps transfer rates for given bandwidth, delay and number of bits. We do so in order not to model the complex effects of TCP/IP for large transfers, in particular the sliding window mechanism. Let $r_{t_{LAT},b}(n)$ be the rate in the look-up table matching closest t_{LAT}, b and n. The run-time of the phase can be computed as

$$t_4 = (n_i + n_t + n_o) \cdot r_{t_{LAT},b}(n_i + n_t + n_o)$$

In phase five B evaluates the circuit at his site and returns A's encrypted outputs. Let t^B_{OWH} denote the time for the operation to perform in order to compute the key of the next gate. Let mtu denote the payload in bits per packet. The total time of phase five can be computed as

$$t_5 = n_g t^B_{OWH}(\kappa) + \left\lceil \frac{\beta^A l \kappa}{mtu} \right\rceil t_{LAT}$$

The overall run-time for the GC based SC is the sum of all phases, $t_{GC} = \sum_{i=1}^{5} t_i$.

3) HE based SC: In the SC sub-model based on HE we combine the protocols of [10], [13], [25] and additive sharing in order to implement the operations in O. All operations are then executed sequentially updating the respective local secret shares.

In order to implement SC on additive secret shares it would suffice to implement multiplication securely. The secure dot product protocol of [13] considers $\vec{x}$ and $\vec{y}$ to be shared between A and B in a way that A knows all elements in $\vec{x}$ while B knows all elements in $\vec{y}$. In order to sequentially compose the operations, we, however, consider additively shared elements, i.e., each element in both vectors is additively shared. That is, A knows $\vec{x}^A = \langle x_{A,1}, \cdots, x_{A,n} \rangle$ and $\vec{y}^A = \langle y_{A,1}, \cdots, y_{A,n} \rangle$ while B knows $\vec{x}^B = \langle x_{B,1}, \cdots, x_{B,n} \rangle$ and $\vec{y}^B = \langle y_{B,1}, \cdots, y_{B,n} \rangle$. We therefore use a slightly modified variant. (For details we kindly refer to the full version of this paper).

Some operations of the algorithm can be translated into local operations while others have to be handled by sub-protocols. The operations in the HE based SC sub-model are performed in a field of size k_{HE} bits.

At player ρ's site local operation $\oplus$ requires time $t^{\rho}_{ADD}(k_{HE})$. Local operation $\ominus$ is identical to $\oplus$ in a field, hence requires the same time. Local operation $\odot_c$ requires time $t^{\rho}_{MUL}(k_{HE})$. Local operation $\ll$ is identical to $\odot_c$, i.e., requires the same time.

Local operations specified in O are executed by both parties simultanously. The time the operations takes therefore is bound by the slower party. Hence,

$$t_{\oplus} = \max(t^A_{ADD}(k_{HE}), t^B_{ADD}(k_{HE}))$$

and

$$t_{\odot_c} = \max(t^A_{MUL}(k_{HE}), t^B_{MUL}(k_{HE}))$$

The remaining operations $\odot_\epsilon$, $\oslash_c$, $\leq$ and $\gg$ are implemented by cryptographic sub-protocols. The run-time of the operation includes time for local computation and communication. The protocols for $\odot_\epsilon$ by [13], $\oslash_c$ by [10] and $\leq$ by [25] differ from the one of Yao [11] in their relation between computation and communication. Yao's protocol [11] performs comprehensive computation (encryption), then transmits a relatively (compared to the other protocols) large set of data at one point in time, and finally performs again comprehensive computation (evaluation). The considered HE based operations exhibit a different behavior w.r.t. network usage; multiple small messages are transmitted between the players rather then a single chunk. We therefore compute the communication time based on packets and latency rather than bandwidth and transfer rate. Let $t^{\rho}_{ENC}(k_{HE})$ be the time ρ requires for one encryption in the HE scheme, $t^{\rho}_{DEC}(k_{HE})$ the time of one decryption and $t^{\rho}_{POW}(k_{HE})$ the time of one exponentiation in the field of the modulus of the HE scheme.

Let mtu denote the payload in bits per packet. We compute from the protocol

$$\begin{aligned} t_{\odot_\epsilon} \quad = \quad & 2\epsilon \left(t^A_{ENC}(k_{HE}) + t^B_{POW}(2k_{HE})\right) + t^B_{ENC}(k_{HE}) + \\ & t^A_{DEC}(k_{HE}) + \left(\left\lceil \frac{2\epsilon k_{HE}}{mtu} \right\rceil + 1 \right) t_{LAT} \end{aligned}$$

For operation $\leq$ we use the protocol from [25] which presents the fasted HE based solution to the problem. Furthermore, it enables us to use the same HE scheme as for all other operations. The output of the protocol in its original version is a random number which is either positive or negative depending on $a \leq b$ being $true$ or $false$. With an additional step of computation and communication it is possible to output an additively shared 0 or 1. We compute

$$\begin{aligned} t_{\leq} \quad = \quad & 3t^A_{ENC}(k_{HE}) + 2t^A_{DEC}(k_{HE}) + 5t^B_{ENC}(k_{HE}) + \\ & 3t^B_{POW}(2k_{HE}) + 5t_{LAT} \end{aligned}$$

For operation $\oslash_c$ we use the protocol of [10]. An additively shared value is divided by a constant by dividing both shares by the constant and checking whether a possible wrap-around in the corresponding field has occurred. The overflow can be determined by using an 1-out of-2 OT. Furthermore it requires one $\leq$ operation in order to correct potential rounding error. We compute

$$t_{\oslash_c} = t_{OT} + t_{\leq}$$

Operation $\gg$ can be reduced to $\oslash_c$, i.e., requires the same time.

D. System

The system layer captures the parameters representing the run-time of local computations and network communication.

We estimate these parameters by benchmarking them on the actual systems the SC will be performed. There are operations that are called infrequently while others, like encryption in the GC, are called many times in batch mode. Unfortunately, the overall run-time of the batch execution is not linear in the number of calls due to run-time optimization effects on compiler (JIT) and processor (cache) level [27], [28]. The average per operation is relatively high if the batch of the benchmark is relatively small. It decreases with increasing size of the batch. A bound of n_c iterations exists after which the average changes only unnoticably. We recommend for t^{ρ}_{OWH} to benchmark the time for n_c calls. The actual number of calls is greater than n_c even for simple circuit constructions [29], [8].

A benchmark then measures the overall time between the first and after the last operation. The time of a basic operation parameter is the arithmetic average.

For all local operations not operating in a batch mode, we propose to perform a fixed low number of iterations (e.g., 100) and assigning the median in order to receive an accurate representative.

The system layer also captures the settings of the communication network. These are the network packet delay t_{LAT} and the transfer rate $r_{t_{LAT},b}(n)$. The transfer rate depends on t_{LAT} and b and varies with the number of bits to transfer n. The network parameter mtu captures the size of the payload per packet in bits and can be inferred from the network setup. For small, bursty data transfers we use mtu to determine the number of packets and then measure the time for transmitting each packet. For large data transfer we use the transfer rate to estimate the transmission time.

For benchmarking the network parameters we first perform a *ping* procedure. We send *one* network packet to the other computer who echoes back this *one* network packet. This way we can measure the time for a round-trip which is roughly $2t_{LAT}$.

We then determine the transfer rate $r_{t_{LAT},b}(n)$ for the current network setup. We propose the following approach. A simple look-up table can be created by sending a staggered set of values for n. The set of values needs to be chosen carefully preferably adapted to the considered algorithms, since the spectrum of the look-up table and the time consumed for calibration evidently are a tradeoff.

IV. Accuracy Evaluation

A. Experiment Setup

In this section we present an accuracy evaluation of the PM. We measure the consumed run-time of an actual implementation and compare it against the forecast made by the PM. We ran a large number of SCs covering the range of parameters.

We present the choice of values for the single parameters layer by layer. Tab. III gives an overview of the parameter values.

Parameter	Values	Metric
l	$\{8, 16, 32\}$	bit
Algorithm	LD, NR	
κ_{OT}	$\{128, 160, 192\}$	bit
κ_{GC}	$\{160, 256\}$	bit
κ_{HE}	$\{768, 1024, 1536\}$	bit
b	$\{0.5, 1, 5, 10, 100\}$	Mbit/s
t_{LAT}	$\{0, 12.5, 25, 50, 75\}$	ms
mtu	1500	bytes

TABLE III
PARAMETERIZATION FOR EXPERIMENTS

We first consider input/output and algorithm layer. We aim to cover the various operations of the algorithm layer. We first fix the basic functionality to secure two-party division – a common problem for several SC problems. We then select two algorithms for division for $l \in \{8, 16, 32\}$ bits inputs (resp., outputs) each. The two algorithms are long division (LD) and Netwon-Raphson approximation (NR). Tab. IV gives an overview on the quantity of operations in the algorithm layer required by those algorithms.

Operation	LD	NR
$\oplus$	$(2B+1)\,\delta - 1$	
$\ominus$	$(2B+1)\,\delta - 2$	γ
$\odot_{\epsilon=1}$	$(2B-2)\,\delta$	$2\gamma + 1$
$\odot_c$		γ
$\oslash_c$		$\gamma + 1$
$\ll$	$4\delta - 1$	
$\gg$	2δ	
$\leq$	$B\delta - 1$	

TABLE IV
OVERVIEW ON THE QUANTITY OF OPERATIONS REQUIRED TO PERFORM DIVISION ALGORITHM (IN RELATION TO BASIS B, NUMBER OF DIGITS $\delta = \log_B \left(2^l - 1\right)$ AND γ ITERATIONS OF APPROXIMATION INVARIANT)

In the SC layer we have two SC protocols: GC and HE. We receive by cross-combination four basic SCs: LDGC, LDHE, NRGC and NRHE.

From the initial value we can derive the number of required iterations. Let $\{l \to \gamma\}$ denote the pair of input-length l and number of iterations γ. For an initial value 2 we receive $\{8 \to 12\}$, $\{16 \to 28\}$ and $\{32 \to 60\}$.

LDGC and NRGC only differ in the number of gates of the circuit. Using sub-circuits described in [8], we receive circuits with 1325, 5461 and 22181 gates for LDGC with input-lengths of 8, 16 and 32 bits. In contrast, we receive 132977 gates for NRGC with input-length of 8 bits. Consequently, the run-time for NRGC with 16 and 32 bits input-lengths is infeasibly long and therefore we only conduct experiments with $l = 8$ bits for NRGC.

We use different parameters for the key-lengths. For OT we use $\kappa_{OT} \in \{128, 160, 192\}$ bits and SHA1 as one-way hash function. Depending of the one way hashing algorithm we have different key-lengths for κ_{GC}. Using one-way hashing functions SHA1 we have $\kappa_{GC} = 160$ bits and using SHA2-256 we have $\kappa_{GC} = 256$ bits. For HE we use key-lengths 768, 1024 and 1536 bits.

In the system layer we use two identical, real computer systems connected by a network. In order to simulate different computation vs. communication tradeoffs we vary the network performance via a WAN simulator. We simulate various bandwidths (0.5, 1, 5, 10, 100 Mbit/s) and latencies (0, 12.5, 25, 50, 75 ms) representing typical values for LAN and WAN connections. We leave the MTU to its Ethernet default value of $mtu = 1500$ byte. The timings of basic operations in the system layer are obtained via our benchmark routine.

We implement the four SCs in a programming language for cryptographic protocols, L1 [19]. The L1 compiler translates source code written in L1 language into Java source code. In one measurement we execute two Java programs which communicate with each other using plain TCP/IP sockets. As hardware we use servers hosting four AMD Opteron 885 dual-core 64-bit CPUs and 16 GB RAM.

The machines are linked by a 1 Gbit/s connection to an intermediate PC, which runs Dummynet [30]. Dummynet is a WAN network emulator which allows us to emulate network latency and bandwidth.

We parameterize the PM according to the values depicted in Tab. III and the values received by the benchmark routine. Applying a permutation of all parameters in Tab. III we end up with 600 settings for the GC and 1350 for the HE based protocol, i.e., 1950 in total. In order to receive statistically valid results, we perform 25 runs per setting, i.e., 48750 tests.

B. Experimental Results

We group analysis into two parts: results related to GC and those related to HE.

1) Results for GC: We show in Fig. 1 for each experimental setting the ratio between run-time forecast of the PM and measured absolute run-time of the experiment. The x-Axis contains all 600 experimental settings for GC, grouped by bandwidth b. Regarding Tab. III, an interval of 120 entries belongs to one value of b. Each bandwidth interval is sub-grouped into five sub-intervals, one for each latency t_{LAT}, i.e., 24 experiment settings per sub-interval.

We conclude from Fig. 1 that the PM deviates from the measured absolute run-time values in a range of -27.6% until $+9.5\%$. The bandwidth has the foremost influence on the error. We receive a smaller error range from -14.1% until $+3.7\%$ for lower bandwidths (e.g., $b = 0.5$ Mbit/s), while we receive a broader range from -27.6% until 9.5% for higher bandwidths (e.g., $b = 100$ Mbit/s).

Moreover, for bandwidth values $b \geq 5$ the error spreads most for small latency values t_{LAT}. Eventually, for highest bandwidth $b = 100$ Mbit/s and lowest latency $t_{LAT} = 0$ ms the error of the PM spreads most (-27.6% until $+9.5\%$). t_{LAT} is the second most relevant parameter determining the error range.

This is further supported by Fig. 2 which shows the average of the absolute value of the relative error per bandwidth and latency. One can see from the diagram that the error spreads more (i.e., its absolute value is higher) the higher the bandwidth and the lower the latency is.

The accuracy of the PM gets noticeably better considering WAN values, e.g., connections having arbitrary bandwidth values from Tab. III and $t_{LAT} \geq 25$ ms. The error for such WAN cases spreads between $-14,4\%$ and $+5,6\%$.

Finally, we show by Fig. 3 the correlation between the relative error and the absolute run-time of the SC. The figure shows that the error is particularly high for small run-times. The error becomes smaller the higher the run-time.

2) Results for HE: The results for HE-based experiments are slightly different.

Similar to the GC, we show in Fig. 4 the relative value between the PM forecast and the measured run-time. Analogously to Fig. 1, the diagram is grouped into bandwidth

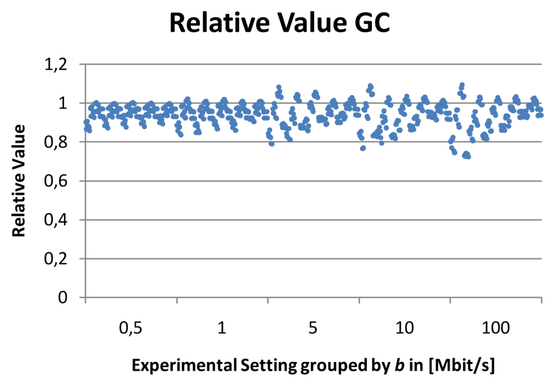

Fig. 1. Rel. value between run-time forecasts and measured run-time of GC-based experiments - grouped by bandwith

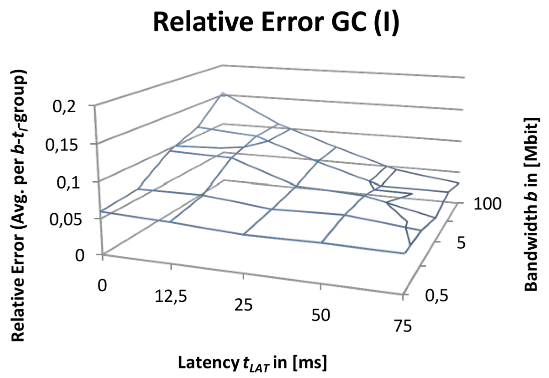

Fig. 2. Avg. per t_{LAT}-b-group of rel. error between run-time forecasts and measured run-time of GC-based experiments

intervals. Regarding Tab. III, an interval of 270 entries belongs to one bandwidth b. Again, each bandwidth interval is grouped into five sub-intervals, one for each latency t_{LAT}, i.e., 54 experiment settings per sub-interval. From Fig. 4 we see that the PM spreads in a range of -14.6% until $+17.9\%$. In terms of largest deviation, the diagram reveals that for small latency values (e.g., $t_{LAT} \leq 25$) ms there are a number of outliers. More precisely, considering the metric of absolute value of relative error for $t_{LAT} = 0$ ms we have a value of 26.8%, while for all other latency values the error is in a range from 18.1% until 21.3%. Considering WAN scenarios (e.g., connections with $t_{LAT} \geq 25$) ms, the PM has an error in a range of -14.6% until $+7.6\%$.

In contrast to the GC-based settings, bandwidth does not play an equally important role with respect to the overall accuracy of the PM. Looking at Fig. 5 underlines this assessment. Fig. 5 is constructed similar to Fig. 2, i.e., it depicts the average per bandwidth and latency of the absolute value of the relative error. Opposite to GC, we cannot determine a significant development of the error depending on bandwidth or latency. Therefore, latency turns out to be most relevant parameter. That is not surprising, since the HE sub-model realizes operators of the PM by cryptographic sub-protocols which frequently exchange smaller messages. For such a

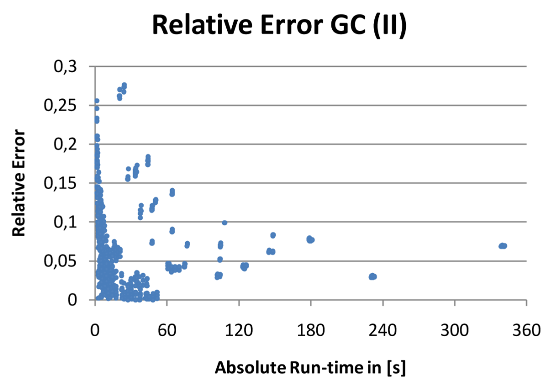

Fig. 3. Rel. value between run-time forecasts and measured run-time of GC-based experiments - sorted by run-time

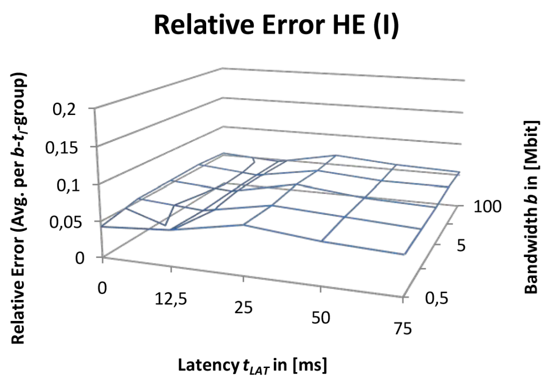

Fig. 5. Avg. per t_{LAT}-b-group of rel. error between run-time forecasts and measured run-time of GC-based experiments

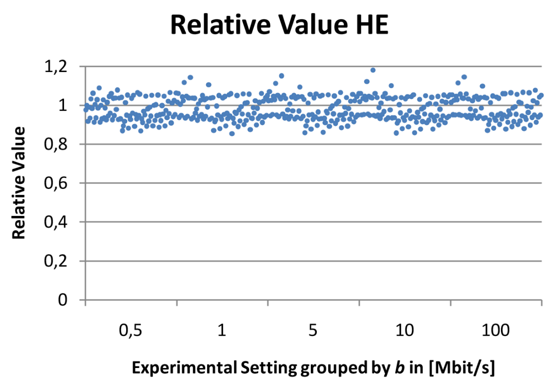

Fig. 4. Rel. error between run-time forecasts and measured run-time of HE-based experiments - grouped by bandwidth

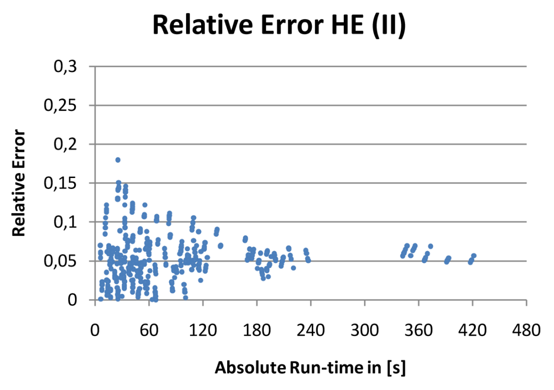

Fig. 6. Rel. error between run-time forecasts and measured run-time of HE-based experiments - sorted by run-time

communication pattern, latency of transmitting a message is highly relevant (rather than the bandwidth). We leverage the fact that modeling network packet transmission is easier and more accurate than modeling small local operations. The more packets are to be transmitted and the higher the latency is, the more accurate is the forecast for a cryptographic protocol.

We show in Fig. 6 also for HE the correlation of forecast error and run-time. For large run-time values, the error converges to a small band of 2.5% to 7.5%.

The evaluation showed that our PM is accurate. The PM is in particular accurate for practical latencies ($t_{LAT} \geq 25$) ms exposing an overall deviation of not more than 22.2%.

V. Discussion

We next discuss some predictions made possible by the PM. We try to answer the following questions

- Which algorithm and SC protocol performs best?
- What is the security trade-off?
- What is more important – network or computation performance?
- What is the impact of unbalanced devices?

A. *Algorithm/SC combination*

We first want to figure out which algorithm and SC yields the best option. There are four options for secure division: LDGC, LDHE, NRGC and NRHE. We consider the exemplary parameter setting $l = 8$ bits, $b = 5$ Mbit/s and $t_{LAT} = 50$ ms. Key-lengths are set to $k_{OT} = 160$ bits, $k_{GC} = 128$ bits and $k_{HE} = 1024$ bits. Fig. 7 shows the run-time forecasts of our PM. For the selected parameters LDGC represents the best choice, followed by NRHE. LDHE and NRGC are far behind.

We can conclude that is not possible to solely determine a best algorithm – LD or NR, since using GC for LD is faster than NR and using HE for NR is faster than LD. We can also conclude that is not possible to solely determine a best SC protocol – GC or HE, since for the LD algorithm GC is faster than HE and for the NE algorithm HE is faster than GC. Performance comparison makes only sense for a combination of algorithm and SC protocol.

We next investigate the impact of the input-length for LDGC and NRHE – our two best combinations – setting $l \in \{8, 16, 32\}$. The diagram in Fig. 8 shows, that LDGC outperforms NRHE for $l \in \{8, 16\}$. For $l = 32$, however, NRHE performs noticeably better than LDGC.

We can conclude that between the two best options – LDGC and NRHE – there is no always best option. In this case the better computation complexity of NRHE makes the predicted difference for larger inputs. As input-lengths increase NRHE will outperform LDGC.

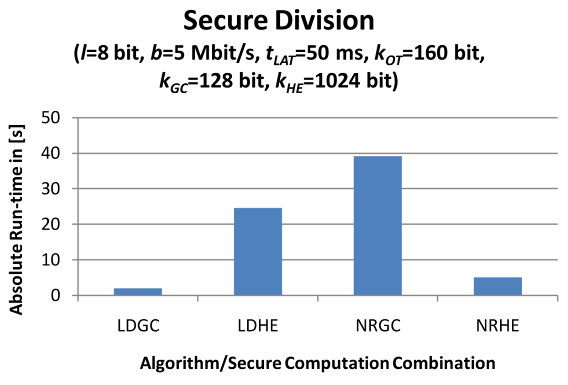

Fig. 7. Run-time of secure division with $l = 8$ bits, $k_{HE} = 1024$ bits, $k_{GC} = 128$ bits, $b = 5$ Mbit/s and $t_{LAT} = 50$ ms.

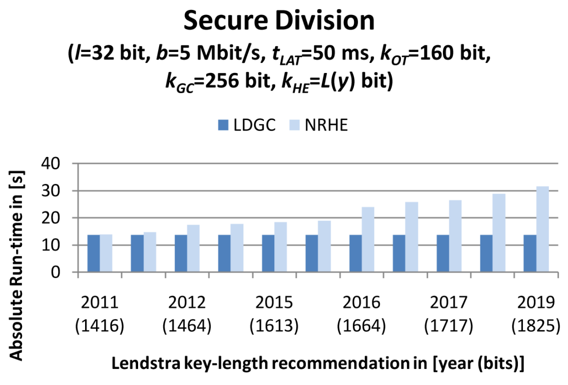

Fig. 9. Run-time of secure division with $l = 32$ bits, $k_{GC} = 256$ bits, $b = 5$ Mbit/s and $t_{LAT} = 50$ ms.

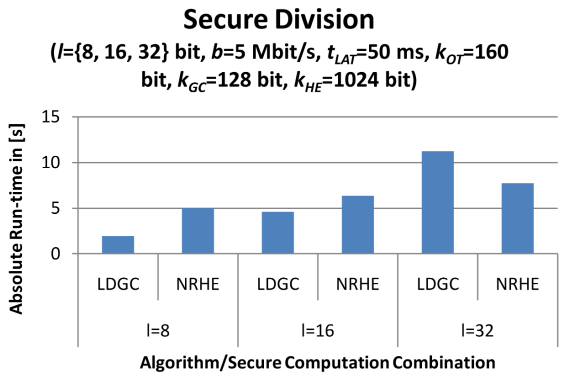

Fig. 8. Run-time of secure division with $l \in \{8, 16, 32\}$ bits, $k_{HE} = 1024$ bits, $k_{GC} = 128$, $b = 5$ Mbit/s and $t_{LAT} = 50$ ms.

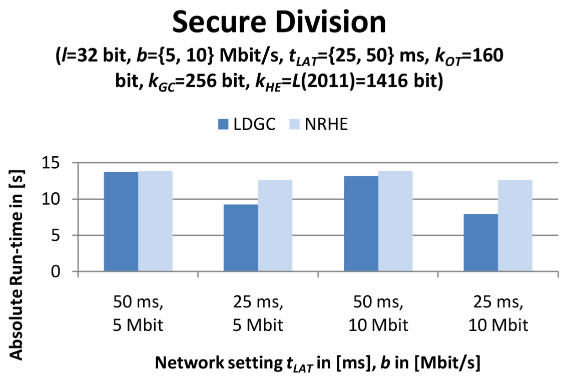

Fig. 10. Run-time of secure division with $l = 32$ bits, $k_{HE} = 1416$ bits, $k_{GC} = 128$ bits, $b = \{5, 10\}$ Mbit/s and $t_{LAT} = \{25, 50\}$ ms.

B. Security

Determining the security of encryption is a parameter of the key-length. The longer the key, the longer the security is supposed to last. Furthermore, with increasing computational performance key-lengths are required to increase as well in order to withstand computational attacks. In [31] key-lengths were recommended for a security horizon for the coming years. We vary the key-length parameters to assess the performance within the next ten years.

We can only set the key-length k_{HE} of Paillier's HE system to the recommendations in [31]. We cannot simply set intermediate values for k_{GC}, since this would require modifying the hashing algorithm. We rather set the hash-length to $k_{GC} = 256$ bits by using SHA-256, by far exceeding the recommendation.

We fix the other parameters to $l = 32$ bits, $b = 5$ Mbit/s and $t_{LAT} = 50$ ms. Fig. 9 shows the development of run-time depending on the recommended key-length. For the first years, starting with 2010, LDGC and NRHE yield almost identical run-times. Following recommendations for 2016, LDGC clearly performs over NRHE. Considering the former Fig. 8 (which represents by $k_{HE} = 1024$ bits a recommendation for 2002) NRHE could once outperform LDGC - following recommendations after 2015, LDGC outperforms NRHE.

We can conclude: As security requirements rise LDGC will outperform NRHE. Furthermore, since security requirements are to rise with computational performance, we can also conclude: As computational performance increases LDGC will outperform NRHE.

C. Network

A common question in judging SC performance is whether computation or communication complexity is more important. Using our PM we can make an integrated forecast and determine whether a performance is rather computation or communication bound.

Keeping the setting of $l = 32$ bits and $k_{HE} = 1416$ bits (matching the recommendation for the year 2011), we next vary the network performance. Both, computation and network performance are likely to increase over the coming years. The difficult question is to predict which one will rise faster.

We simply assume – without further research, guarantee or loss of generality – that network performance is to increase faster and improve the corresponding parameters while keeping computation parameters fixed. We consider an improvement of $t_{LAT} = 50$ ms to 25 ms and $b = 5$ Mbit/s to 10 Mbit/s.

Fig. 10 depicts four cases, using fixed values $l = 32$ bits, $k_{HE} = 1024$ bits and $k_{GC} = 128$ bits: the baseline setting with $t_{LAT} = 50$ ms and $b = 5$ Mbit/s, a decrease of latency only to $t_{LAT} = 25$ ms, an increase of bandwidth only to $b = 10$ Mbit/s and the best case with decrease of latency and increase of bandwidth. Fig. 10 shows that the influence on LDGC and NRHE differs. LDGC reduces run-time noticeably by reducing the latency and slightly by raising bandwidth. NRHE slightly reduces run-time only for reduced latency.

We can conclude: If network performance is to rise faster than computation performance, LDGC will outperform NRHE.

D. Heterogeneous Devices

We finally consider the impact of heterogeneous devices, i.e., settings with P_A and P_B running devices with different computational power. The considerations can be used to investigate the effect of changing usage patterns, e.g., from scenarios between enterprise informations systems (i.e., computational powerful servers) to scenarios with mobile users running netbooks or smartphones. It may also be used to simulate effects of the development of computational power using battery optimized (weak) devices.

We consider the same setup as for Fig. 8, i.e., $l \in \{8, 16, 32\}$ bits, $t_{LAT} = 50$ ms, $b = 5$ Mbit/s, $k_{HE} = 1024$ bits and $k_{GC} = 128$ bits. We first adapt one player to run a netbook rather than a server. We use a netbook a which is equipped with a 1.6 GHz N270 Intel Atom CPU and 1 GB RAM.

Considering the setup of two differently computational strong devices, we recommend P_A to run the faster device. The reason is that the GC protocol requires more computational work to be done (i.e., encryption of each truthtable entry) by the first player.

Comparing the diagram in Fig. 11 to the one in Fig. 8, run-times of LDGC and NRHE change non-uniformly. While LDGC does not expose any noticeable change, NRHE remarkably increases. More precisely, the run-time of NRHE increases by a factor of 10 for all values of l.

The conclusion is difficult to draw, but the CPU architecture seems to have a significant impact on SC performance.

Finally, we have both players run a netbook. We present in Fig. 12 the corresponding forecast. Comparing the results from the diagram to those of Fig. 8 (both players running servers), we see that LDGC does not noticeably change run-time. NRHE, however, further extends the former factor of 10 to 20, for all values of l.

This underpins our conclusion that the CPU architecture has significant impact on SC performance.

VI. Comparison to Complexity

Comparing the output of the PM against asymptotic complexity measures, we obtain the results depicted in Tab. V.

Algorithm/ Protocol	Computation Exp.	Computation OWH	Communication	Rounds
LDGC	$\mathcal{O}(l)$	$\mathcal{O}(l^2)$	$\mathcal{O}(l^2)$	$\mathcal{O}(1)$
NRGC	$\mathcal{O}(l)$	$\mathcal{O}(l^3)$	$\mathcal{O}(l^3)$	$\mathcal{O}(1)$
LDHE	$\mathcal{O}(l)$	$\mathcal{O}(l)$	$\mathcal{O}(l)$	$\mathcal{O}(l)$
NRHE	$\mathcal{O}(l)$	$\mathcal{O}(l)$	$\mathcal{O}(l)$	$\mathcal{O}(l)$

TABLE V
Asymptotic complexities

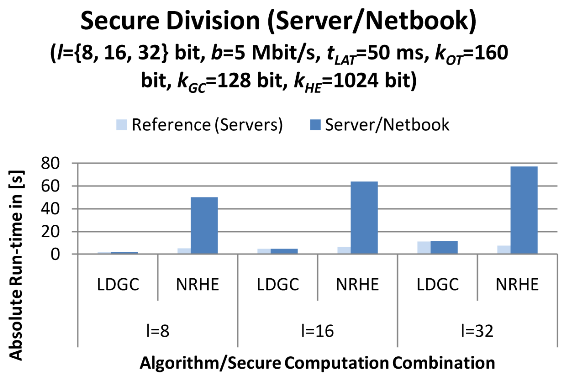

Fig. 11. Absolute run-time of secure division SCs with $k_{HE} = 1024$ bits, $k_{GC} = 128$ bits, $b = 5$ Mbit/s and $t_{LAT} = 50$ ms.

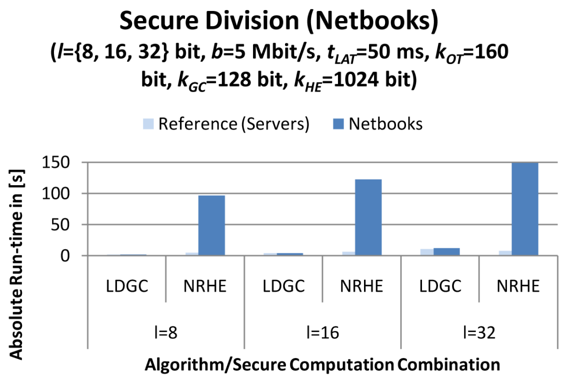

Fig. 12. Absolute run-time of secure division SCs with $k_{HE} = 1024$ bits, $k_{GC} = 128$ bits, $b = 5$ Mbit/s and $t_{LAT} = 50$ ms.

As already mentioned in the introduction, in theoretical analysis cryptographic protocols are assessed w.r.t. their compuation, communication and round complexities. Computation complexity is sometimes only assessed for the most costly operations of modular exponentiations disregarding less costly operations, such as OWH functions.

The complexities depicted in Tab. V are reflected in the PM and are composed as follows. GC requires α^B 1-out of-2 OTs. Using the OT protocol of [26], each OT requires a constant number of OWH calls and exponentiations, as well as a constant number of messages of constant size. The computation and communication complexity therefore is $\mathcal{O}(1)$ for a single OT, and $\mathcal{O}(l)$ for the OTs alltogether. The transmission of all OTs can be done in parallel such that the round complexity is $\mathcal{O}(1)$. Preparing and evaluating the circuit requires $\mathcal{O}(n_g)$ OWH calls (refering to Tab. II that is $\mathcal{O}(l^2)$ for LD and $\mathcal{O}(\gamma \cdot l^2) = \mathcal{O}(l^3)$ for NR), as well as transmission of $\mathcal{O}(n_g)$ bytes in a constant number of rounds. Communication complexity therefore is identical to computation complexity and round complexity is $\mathcal{O}(1)$.

For the sub-protocols used in HE based SC we observe the following complexities. Each respective operation shown in Tab. IV requires $O(1)$ number of exponentiations, communication and rounds (if $\epsilon = O(1)$). LD performs $\mathcal{O}(\delta)$ and NR performs $\mathcal{O}(\gamma)$ such operations. Since $\delta = \mathcal{O}(l)$ and $\gamma = \mathcal{O}(l)$, we obtain the depicted complexities of $\mathcal{O}(l)$ for LDHE and NRHE.

In Tab. V we have normalized all complexities to the input length l. We conclude that the HE based SCs have lower computation and communication complexity than the GC variants. However, the opposite is true w.r.t. round complexity. w.r.t. to the algorithms there are similar problems: In the HE protocols LD and NR have the same complexity, while in the GC model LD is less complex than NR. We already see that a conclusive decision which is the best protocol is not feasible from the complexity measures.

VII. Conclusion

We introduced a PM for secure two-party computation. We empirically evaluated the accuracy of our PM. We further discussed several predictions made possible by the PM considering secure division as a potential use-case. We demonstrated varying several parameters in the PM how difficult it can be to select the best SC. The forecasts made by our PM can help to make informed decisions, e.g. in an optimizing compiler. Our PM integrates various complexity measures, such as computation, communication and round complexity. It furthermore provides constants for its terms resulting in a run-time forecast.

Considering the use-case of secure devision, an assessment for future development remains hard. Clearly, LDGC and NRHE represent the best options, but there are conflicting future trends, e.g. for input-length and key-length. Considering our current parameters we choose LDGC as the best option, today and in the foreseeable future, for the JELS application of [8].

The conclusion of our PM compared to theoretical complexity measures highlights its benefits. Without exposing the hidden constants, it is not feasible to determine the best of the two candidates LDHE and NRHE. Moreover, the situation is significantly complicated in the diverse system environments our PM caters for.

Our work therefore underpins that a PM enables improved performance assessment of secure computations in practice.

References

[1] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, “Fairplay - a secure two-party computation system,” in *Proceedings of the USENIX security symposium*, 2004, pp. 287–302.

[2] A. Ben-David, N. Nisan, and B. Pinkas, “Fairplaymp: a system for secure multi-party computation,” in *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 257–266.

[3] Virtual Ideal Functionality Framework, “http://www.viff.sk.”

[4] Sharemind, “http://sharemind.cs.ut.ee/wiki/.”

[5] W. Henecka, S. K ögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, “Tasty: tool for automating secure two-party computations,” in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 451–462.

[6] J. Bethencourt, “http://acsc.cs.utexas.edu/libpaillier/.”

[7] F. Kerschbaum, D. Dahlmeier, A. Schröpfer, and D. Biswas, “On the practical importance of communication complexity for secure multi-party computation protocols,” in *SAC*, 2009, pp. 2008–2015.

[8] R. Pibernik, Y. Zhang, F. Kerschbaum, and A. Schröpfer, “Secure collaborative supply chain planning and inverse optimization – the jels model,” in *European Journal of Operational Research, to appear*, 2010.

[9] P. Bunn and R. Ostrovsky, “Secure two-party k-means clustering,” in *Proceedings of the 14th ACM conference on Computer and communications security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 486–497.

[10] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara, “Private collaborative forecasting and benchmarking,” in *Proceedings of the ACM Workshop on Privacy in an Electronic Society*, 2004, pp. 103–114.

[11] A. Yao, “How to generate and exchange secrets,” in *In Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986, pp. 162–167.

[12] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *Proceedings of EUROCRYPT, Lecture Notes in Computer Science 1592*, 1999, pp. 223–238.

[13] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen, “On private scalar product computation for privacy-preserving data mining,” in *7th International Conference on Information Security and Cryptology*, 2004.

[14] P. Bogetoft, D. L. Christensen, I. Damgard, M. Geisler, T. Jakobsen, M. Kroigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft, “Secure multiparty computation goes live,” in *13th International Conference on Financial Cryptography and Data Security*, 2009.

[15] F. Kerschbaum, “Practical privacy-preserving benchmarking,” in *23rd IFIP International Information Security Conference*, 2008.

[16] B. Pinkas, T. Schneider, N. Smart, and S. Williams, “Secure two-party computation is practical,” in *Advances in Cryptology ASIACRYPT 2009*, ser. Lecture Notes in Computer Science, M. Matsui, Ed. Springer Berlin / Heidelberg, 2009, vol. 5912, pp. 250–267.

[17] O. Catrina and C. Dragulin, “Multiparty computation of fixed-point multiplication and reciprocal,” in *DEXA Workshops*, 2009, pp. 107–111.

[18] E. Kiltz, G. Leander, and J. Malone-Lee, “Secure computation of the mean and related statistics,” in *Proceedings of Theory of Cryptography Conference, Lecture Notes in Computer Science 3378*, 2005, pp. 283–302.

[19] A. Schröpfer, F. Kerschbaum, and G. Müller, “L1 - an intermediate language for mixed-protocol secure computation,” in *Proceedings of the 34th Annual IEEE International Computer Software and Applications Conference, COMPSAC*, 2011.

[20] R. Sion and B. Carbunar, “On the computational practicality of private information retrieval,” in *In Proceedings of the 14th ISOC Network and Distributed Systems Security Symposium*, 2007.

[21] D. Beaver, S. Micali, and P. Rogaway, “The round complexity of secure protocols,” in *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, 1990, pp. 503–513.

[22] M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault tolerant distributed computation,” in *Proc. of 20th ACM Symposium on Theory of Computing (STOC)*, 1988, pp. 1–10.

[23] R. Cramer, I. Damgard, and U. Maurer, “General secure multi-party computation from any linear secret sharing scheme,” in *Eurocrypt*, 2000.

[24] T. Reistad and T. Toft, “Linear, constant-rounds bit-decomposition,” in *ICISC*, 2009, pp. 245–257.

[25] F. Kerschbaum, D. Biswas, and S. de Hoogh, “Performance comparison of secure comparison protocols,” 2009, pp. 133 –136.

[26] M. Naor and B. Pinkas, “Efficient oblivious transfer protocols,” in *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001, pp. 448–457.

[27] J. Henning, “Spec cpu2000: measuring cpu performance in the new millennium,” *Computer*, vol. 33, no. 7, pp. 28 –35, 2000.

[28] S. Pieper, J. Paul, and M. Schulte, “A new era of performance evaluation,” *Computer*, vol. 40, no. 9, pp. 23 –30, sep. 2007.

[29] I. Wegener, *Effiziente Algorithmen für grundlegende Funktionen*. Teubner, 1996.

[30] M. Carbone and L. Rizzo, “Dummynet revisited,” *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, 2010.

[31] A. K. Lenstra and E. R. Verheul, “Selecting cryptographic key sizes,” *J. Cryptology*, vol. 14, no. 4, pp. 255–293, 2001.