Automatic Protocol Selection in Secure Two-Party Computations
(Full Version)[⋆]

Florian Kerschbaum[1], Thomas Schneider[2], and Axel Schröpfer[1]

[1] SAP AG
{florian.kerschbaum,axel.schroepfer}@sap.com
[2] TU Darmstadt
thomas.schneider@ec-spride.de

Abstract. Performance of secure computation is still often an obstacle to its practical adaption. There are different protocols for secure computation that compete for the best performance. In this paper we propose *automatic protocol selection* which selects a protocol for each operation resulting in a mix with the best performance so far. Based on an elaborate performance model, we propose an optimization algorithm and an efficient heuristic for this selection problem. We show that our mixed protocols achieve the best performance on a set of use cases. Furthermore, our results underpin that the selection problem is so complicated and large in size, that a programmer is unlikely to manually make the optimal selection. Our proposed algorithms nevertheless can be integrated into a compiler in order to yield the best (or near-optimal) performance.

Keywords: Secure Two-Party Computation, Performance, Optimization, Protocol Selection

1 Introduction

Secure two-party computation allows two parties to compute a function f over their joint, private inputs x and y, respectively without revealing their private inputs or relying on a trusted third party. Afterwards, no party can infer anything about the other party's input except what can be inferred from her own input and the output $f(x, y)$. Secure computation has many applications, e.g., in the financial sector, and has been successfully deployed in commercial and industrial settings [8,29,7].

Performance is still often an obstacle to practical adoption of secure computation, even in the widely used semi-honest security model. A number of protocols compete for the best performance in this model. Recently, the garbled circuit implementation FastGC [24] has been used in several privacy-preserving applications, including [22,23,21,45,46], but still garbled circuits have some inherent limitations, e.g., due to the large circuit size of some functionalities such as multiplication. In this paper we propose a different approach. Instead of relying on a single protocol we mix protocols. Then, based on an extended performance model we *automatically* select the best protocol for a sub-operation. In all prior works this selection has been performed manually, e.g., [53,20,25,5]. We present two algorithms for the protocol selection – an optimization based on integer programming and a heuristic. We apply these to three use cases from the literature: secure joint economic lot-size, biometric identification, and data mining. We use the evaluation of their implementation in the intermediate language of [54] to test three hypotheses:

– Our mixed protocols are faster than a pure garbled circuit implementation.
– The results of our heuristic and the optimum found by integer programming are close.
– The protocol selection problem is too complicated to be solved manually by the programmer.

Our heuristic can then be used in a compiler to automatically select the fastest sub-protocols in secure computations.

Our Contributions and Outline. In summary, this paper contributes

– an *extended performance model* for mixed protocol secure computation,
– two *selection algorithms* to automatically select mixed protocols with (near-) optimal performance based on this model,
– an *evaluation* based on three use cases from the literature.

Our paper is structured as follows: In §2 we review related work. In §3 we describe our mixed protocols for secure computation. §4 explains the corresponding cost model including conversion costs. We present our selection algorithms in §5 and our evaluation results in §6. Our conclusions are summarized in §7.

[⋆] An extended abstract of this work was published in [30]; please cite the conference version [31].

2 Related Work

Our results are based on the performance model framework of [54] for forecasting runtimes of secure two-party computations based on garbled circuits and homomorphic encryption. In §4.1 we provide a summary of this framework and extend it in §4.2 to cover conversion between the protocols. For completeness, we note that there are also other techniques for secure two-party computation beyond the ones we cover in this work, e.g., the GMW protocol [19] implemented in the semi-honest setting in [11] and extended to the malicious setting in [44]. However, as this protocol also favors Boolean circuits, but differently from garbled circuits has a non-constant number of rounds, we chose garbled circuits in our work. Furthermore, this and other protocols can be integrated into our main approach by extending the performance model of [54] accordingly.

[28] describes automatic optimizations of secure computation protocols that automatically infer which operations can be performed locally by each party. This approach is orthogonal to ours that automatically selects the most efficient sub-protocol; combining both approaches yields even more efficient protocols.

There are several implementation frameworks for secure computation. Frameworks for secure two-party computation are either based on garbled circuits (e.g., Fairplay [4,39], FastGC [24], VMCrypt [38], and CBMC-GC [21]) or homomorphic encryption (e.g., VIFF [12]). The L1 framework [55] allows to describe secure computation protocols that employ both techniques, garbled circuits and homomorphic encryption. The TASTY framework [20] provides additional support for conversions between these two approaches. Both, L1 and TASTY require to specify which part of the protocol should be run with which technique. We provide the first method to automatically partition a functionality into sub-techniques.

The Sharemind framework [6] implements secure multi-party computation for three players using an additive secret sharing scheme over the ring $\mathbb{Z}_{2^{32}}$. The compiler of [37] implements secure two-party computations expressed using operations in the field $\mathbb{Z}_q$ of integers modulo a prime q and in the multiplicative subgroup of order q in $\mathbb{Z}_p^*$ for $q|p-1$ with generator g. Our protocols use additive secret sharing over $\mathbb{Z}_{2^l}$ among the two players for intermediate values (cf. §3.2).

Several protocols benefit from the combination of homomorphic encryption and garbled circuits, including [53,20,25,5]. In these protocols, the partitioning into sub-protocols was defined manually, whereas our methods allow to automatically find a good partition.

The authors of [41] describe a technique to compile functionalities described in Fairplay's Secure Function Definition Language (SFDL) [4,39] into Boolean circuits in a memory-efficient way. For this, they first compile the SFDL program into an intermediate language that represents operations as three-operand code. As we use a similar three-operand code language to describe the functionality that needs to be computed securely (cf. §4.1), the compiler of [41] could be easily extended to compile SFDL programs into our input language.

3 Secure Computation Protocols

We integrate two protocols for performing secure two-party computations – garbled circuits and homomorphic encryption. Both protocols are generic, i.e., they can securely implement any ideal functionality. Nevertheless they have different performance characteristics as shown by the performance evaluations in [20,54]. Throughout the paper we name the two parties Alice A and Bob B.

Next we explain the two basic protocols in §3.1 and §3.2, give the conversions that allow to combine and automatically select between both protocols in §3.3, and give background on the underlying semi-honest security model in §3.4.

3.1 Garbled Circuits

Garbled circuits, introduced by Yao [56], were the first generic protocol for secure two-party computation. An excellent introduction can be found in [39] which also presents the first implementation of this protocol. For the purposes of this paper a high-level overview without the technical details of encryption suffices.

Yao's garbled circuits protocol allows secure computation of an arbitrary ideal functionality that is represented as a Boolean circuit C. The basic idea is that C is evaluated on symmetric keys where one key corresponds to the plain value 0 and another to the plain value 1. Alice creates for each gate of C an encrypted table such that given the gate's input keys only the corresponding output key can be decrypted. Then, Alice sends to Bob the keys for the input wires of C in an oblivious manner: For each of Bob's inputs, both parties run a 1-out-of-2 oblivious transfer (OT) protocol. The OT protocol ensures that Bob obtains only the key corresponding to his input whereas Alice does not learn Bob's input. Now, Bob can use the encrypted tables to evaluate C under encryption. Finally, Bob sends the keys that correspond to Alice's outputs back to Alice. For his outputs, he is given a mapping that allows him to decrypt the output keys into plain output values.

For Yao's garbled circuits protocol we use the following optimizations and instantiations that are implemented in FastGC [24] (which is used in many recent works such as [22,23,21,45,46]) and VMCrypt [38]: For OT we use OT extensions of [26] with the OT protocol of [42] for the base OTs. For garbled circuits we use free XOR gates [34], garbled row reduction [43,51], and pipelining [24]. All these protocols and constructions are proven secure against semi-honest adversaries based on the random oracle and the computational Diffie-Hellman assumptions.

3.2 Homomorphic Encryption

Secure computation can also be implemented based on additively homomorphic encryption. On the one hand, opposed to fully homomorphic encryption [15], additively homomorphic encryption only implements addition (modulo a key-dependent constant) as the homomorphic operation. On the other hand, additively homomorphic encryption is almost as fast as standard public-key cryptography, whereas the practicality of fully homomorphic encryption schemes is still subject to research, e.g., [16].

Let $E_X(x)$ denote the encryption of plaintext x encrypted under X's (Alice's or Bob's) public key and $D_X(c)$ the corresponding decryption of ciphertext c. Then the additive homomorphism can be expressed as $D_X(E_X(x) \cdot E_X(y)) = x + y$. Multiplication with a constant c can easily be derived as $D_X(E_X(x)^c) = cx$.

Secure computation of an arbitrary functionality represented as arithmetic circuit can be built from homomorphic encryption as follows. Each variable is secretly shared between Alice and Bob. Let x be a variable of bit length l. Then Alice has share x_A and Bob has share x_B, such that $x = x_A + x_B \bmod 2^l$.

In order to securely implement the ideal functionality it suffices to securely implement addition and multiplication of shares. Addition of $x = x_A + x_B$ and $y = y_A + y_B$ (of the same bit-length l) can be implemented locally by addition of each party's shares. Multiplication $z = x \cdot y$ needs to be implemented as a protocol. Let σ be the statistical security parameter in the share conversion protocol of [13] and r be a uniformly random number of bit length $2l + \sigma + 1$. We use the following protocol for secure multiplication of shares:

$A \longrightarrow B$: $E_A(x_A), E_A(y_A)$
$B \longrightarrow A$: $E_A(c) = E_A(x_A)^{y_B} E_A(y_A)^{x_B} E_A(r) = E_A(x_a y_B + y_A x_B + r)$
A: $c = D_A(E_A(c))$, $z_A = x_A y_A + c \bmod 2^l$
B: $z_B = x_B y_B - r \bmod 2^l$.

It is easy to verify that $z_A + z_B = (x_A + x_B)(y_A + y_B) \bmod 2^l$. Also other operations can be implemented using homomorphic encryption (cf. §4.1).

In our implementation we use Paillier's cryptosystem [49] which is secure against chosen plaintext attacks (IND-CPA) under the decisional composite residuosity assumption.

3.3 Conversion

In the following, we describe how secure computations based on garbled circuits and homomorphic encryption can be combined by converting from one representation of intermediate values to the other. Our methods used for these conversions are similar to those of previous works [20,33], but more efficient as we directly compute on l-bit shares instead of computing on ciphertexts with longer masks: In previous approaches, one party held an l-bit value that is additively homomorphically encrypted under the public key of the other party. To convert such a value into an input of a garbled circuit required to add a $(\sigma + l)$-bit mask to the encrypted value, send this ciphertext back, and after decryption take off the $(\sigma + l)$-bit mask in the garbled circuit. Conversion in the opposite direction is similar. In these previous approaches the mask had to be σ bits longer than l in order to statistically hide the l-bit value. In our approach described below we directly combine the shares modulo 2^l and hence do not require expensive masking, decryption, and transfer of the ciphertext.

Homomorphic Encryption to Garbled Circuits. Assume that we want to compute a sub-functionality f using garbled circuits where one of the l-bit inputs x was computed using homomorphic encryption, i.e., x is represented as shares x_A, x_B with $x = x_A + x_B \bmod 2^l$. To use x as input for the garbled circuit, we extend the inputs of the garbled circuit computing f with an l-bit addition circuit to which A provides input x_A and B provides input x_B, i.e., the slightly larger garbled circuit computes $f(\ldots, x_A + x_B \bmod 2^l, \ldots)$. Note that reduction modulo 2^l is easily obtained by dropping the most significant carry bit.

Garbled Circuits to Homomorphic Encryption. Similarly, we can convert the output z of a sub-functionality that has been computed using garbled circuits into secret shares z_A, z_B that can later on be used for secure computations using homomorphic encryption. For this, we extend the output of the garbled circuit with an l-bit subtraction circuit whose subtrahend is a randomly chosen l-bit share z_A provided by A. We modify the garbled circuit protocol such that only B obtains the output $z_B = z - z_A \bmod 2^l$, i.e., he does not send the output keys back to A. Again, reduction modulo 2^l is easily obtained by dropping the most significant carry bit.

Optimization. Note that we only need to convert the inputs and outputs of operations that are securely computed with a different protocol type. Furthermore, each variable needs to be converted at most once and then can be used as input to all sub-functionalities.

3.4 Security

All protocols described in this section—garbled circuits, homomorphic encryption, and mixed protocols—are secure in the semi-honest model. In this model participants follow the protocol as prescribed, but keep a record of the messages received and try to infer as much information as possible about the other party's input [18]. Protocols secure in the semi-honest model ensure that an adversary cannot infer any information beyond what he can infer from its input and output of the protocol. This model covers many real-life threats such as attacks by honest but curious insiders.

For garbled circuits a proof of security can be found in [36]. Proofs for the protocols using homomorphic encryption can be found in [1,17,27]. For security of the mixed protocol we refer to Goldreich's composition theorem [18].

4 Cost Model

In order to choose which operation to implement using which protocol we need to compare their costs. By cost we mean the (wall clock) run-time of the protocol and its communication. Since the protocol can be composed from sub-protocols of both protocol types – garbled circuits and homomorphic encryption – we need to be able to assess their performance while taking care of additional conversion costs. We base our cost model on the model of [54] which can (reasonably) reliably forecast the protocol run-time and communication for both types of protocols. The accuracy of the forecast mainly determines the effectiveness of our approach. We summarize the layers of the cost model in §4.1 and give the costs for conversions in §4.2.

4.1 Layers

The cost model of [54] is divided into four layers. The top three layers are parameterized by the implemented algorithm and security parameters. The lowest layer is parameterized by the performance of the actual systems on which the protocols are deployed. This performance is measured for some basic operations once. Then, different protocols can be compiled. Alternatively, pre-configured costs for representative environments can be chosen by the programmer.

The first layer captures the number of input and output variables of every player, as well as the bit-length of these variables. The second layer captures the algorithm as a sequential list O of operations. An operation $o = \{\boldsymbol{l}, \circ, \boldsymbol{r}\} \in O$ consists of an assigned variable, a left-operand, an operator and a right-operand (3-operand code). All assignments are single static assignments. We adopt the intermediate language of [54] for our selection algorithms.

The intermediate language currently supports the following operations for which secure protocols are given in [1,17,27,33]. Some of these operations leverage the specific advantages of the respective protocol type, i.e., direct access to single bits and shift operations for garbled circuits or arithmetic operations for homomorphic encryption:

- addition $\oplus$
- subtraction $\ominus$
- dot product $\odot_e$
- multiplication by a constant $\odot_c$
- division by a constant $\oslash_c$
- left shift by a constant $\ll_c$
- right shift by a constant $\gg_c$
- less-or-equal $\leq$

All operands are scalars with the exception of dot product which concurrently multiplies vectors of e elements.

The third layer captures the protocol type and their security parameters, i.e., the lengths of keys in garbled circuits, homomorphic encryption, and oblivious transfer. The fourth layer captures the performance of the systems and the network, i.e., the times for performing local operations (e.g., a homomorphic encryption or a hash-function), and network bandwidth and latency.

Given these parameters, a run-time forecast (cost) of the protocol is computed in the respective model. We implement the cost computation using the arithmetic formulas from [54]. Using an empirical evaluation, the authors of [54] show that these formulas estimate the run-time reasonably precisely: for n forecasts f_i and measurements e_i the average error is only $\left|1 - \frac{1}{n}\sum_{i=1}^{n}\frac{f_i}{e_i}\right| = 3.6\%$.

4.2 Conversion Costs

The model of [54] actually distinguishes the two protocol types. We now need to additionally estimate the conversion costs between the two protocols.

Recall that all operations in the intermediate language are represented in 3-operand code (cf. §4.1). Let $a = b \cdot c$ be such a 3-operand operation. As each variable is assigned exactly once (single static assignment), we can use the assigned variable a as a short notation for the operation. There are two cases when we need to consider conversion costs according to the conversions described in §3.3: If a is implemented using homomorphic encryption, but b (or c) is implemented using garbled circuits, then we need to convert b (or c) from their garbled circuit representation into secret shares by adding an input for Bob's random share z_B and extending the garbled circuit with a subtraction circuit. If a is implemented using garbled circuits, but b (or c) is implemented using homomorphic encryption, then we need to convert b (or c) from their representation as secret shares into inputs for the garbled circuit by adding an addition circuit and inputs for the shares. Again, we emphasize that each operand needs to be converted at most once in the entire mixed protocol.

We can then compute the cost of the mixed protocol as the sum of its parts. For the costs of each part implemented as either protocol type we use the formulas of [54] for homomorphic encryption, the improved formula described in §A for garbled circuits, and the conversion costs described above.

5 Optimal Partitioning

Given the cost model described in §4 we can define the problem of an optimal partitioning of the operations into the protocol types. Consider a compiler that translates a programming language into the intermediate language described in §4.1. In order to construct a cost-optimal (i.e., the fastest) protocol it needs to assign each operation of the intermediate language a protocol type, also considering the conversion costs.

We setup the problem formulation as follows. Let the elements x_i correspond to the left hand-side variable assigned in an operation. We denote with $\mathbb{X}$ the set of these elements (variables). The operator mapping function *op* maps x_i to the right hand-side operators of that operation. The cost function $a(x_i)$ corresponds to the costs for computing x_i using garbled circuits and $b(x_i)$ to the costs using homomorphic encryption, respectively. The cost functions $c(x_i)$ and $d(x_i)$ correspond to the costs for converting x_i from homomorphic encryption to garbled circuits and vice-versa, respectively. The set $\mathbb{Y} \subseteq \mathbb{X}$ of instructions will be implemented using garbled circuits; the set $\mathbb{X}\setminus\mathbb{Y}$ using homomorphic encryption. We formally define the problem as follows:

Definition 1 (Problem Definition). Let $\mathbb{X}$ be a set of elements $x_1, \ldots, x_n$; $op(x_i)$ be a function mapping x_i to a set $\mathbb{F}_i \subseteq \mathbb{X}$; and $a(x_i)$, $b(x_i)$, $c(x_i)$, $d(x_i)$ be four cost functions. Find the subset $\mathbb{Y} \subseteq \mathbb{X}$ that optimizes the following cost function

$$\begin{gathered}\textstyle\sum_{\{x|x\in\mathbb{Y}\}} a(x) + \sum_{\{x|x\in\mathbb{X}\setminus\mathbb{Y}\}} b(x) + \\ \textstyle\sum_{\{x|x\in\mathbb{X}\setminus\mathbb{Y},\exists y. y\in\mathbb{Y}, x\in op(y)\}} c(x) + \\ \textstyle\sum_{\{x|x\in\mathbb{Y},\exists y. y\in\mathbb{X}\setminus\mathbb{Y}, x\in op(y)\}} d(x).\end{gathered}$$

There are some restrictions on the function *op* that are not captured in this problem definition. First, the set $\mathbb{F}_i$ is restricted to a size of at most 2 (three operand code). Second, the set $\mathbb{X}$ is ordered and $op(x_i)$ may only include elements $x_{i'}$ that have been computed already, i.e., $i' < i$. Nevertheless, if we solve the general problem we also solve the restricted problem.

A further complication is that the cost functions in the cost model of [54] do not only depend on the individual operation, but also on its neighbors. As such this already complex problem can only be seen as an approximation of the performance model. We address this in §5.1.

Partitioning problems, e.g., graph partitioning, are typically NP-hard, but unfortunately we cannot provide a hardness proof for our specific instance. First, our specific parameters for the maximum sizes of the partitions (almost the entire set) have not yet been proven NP-hard. Second, our restrictions on the function $op(x)$ complicates the reduction. Nevertheless, we conjecture that the problem is NP-hard.

5.1 Integer Programming (IP)

We search for the best solution to the partitioning problem defined above using an optimization algorithm. However, due to the size of the problem (our largest example considered in §6 has 383 operations) an exhaustive search is prohibitive, such that a more efficient approach for optimization is needed. $0,1$-integer programming is a suitable candidate, but we have to consider some non-linear costs.

In $0,1$ integer programming there are variables $\boldsymbol{z}$ for which an assignment is sought which minimizes a linear objective function $c(z)^T\boldsymbol{z}$ subject to certain constraints. In its standard form it is represented as

$$\begin{gathered}\min \boldsymbol{c}^T\boldsymbol{z}\\ A\boldsymbol{z} \leq \boldsymbol{b}\\ \boldsymbol{z} \in \overrightarrow{\{0,1\}}.\end{gathered}$$

For each element x_i in the set of variables $\mathbb{X}$ we add the following three variables to the integer program:

- $z'_i \in \{0,1\}$ indicates whether the operation assigning x_i will be executed using homomorphic encryption (0) or garbled circuits (1).
- $z''_i \in \{0,1\}$ indicates whether the variable x_i needs to be converted from homomorphic encryption to garbled circuits (1) or not (0).
- $z'''_i \in \{0,1\}$ indicates whether the variable x_i needs to be converted from garbled circuits to homomorphic encryption (1) or not (0).

An element x_i is either implemented as garbled circuits or homomorphic encryption. So one variable suffices, but for conversion we need two variables. An element might not be converted at all, but is never converted in both directions. The objective function to be minimized follows directly from this construction:

$$\min\left(\sum_i a(x_i)z'_i - \sum_i b(x_i)z'_i + \sum_i c(x_i)z''_i + \sum_i d(x_i)z'''_i\right).$$

One complication of this objective function is the non-linearity of garbled circuit execution time. As described in [54], side effects on OS and hardware level (like JIT compilation, CPU caching, etc.) lead to non-linear costs per gate if the number of gates is below a certain threshold. These effects have an influence on the cost objective of the integer program. Sums of costs for single garbled circuits of adjacent operations of the SSA algorithm are likely (due to their small size) to be higher than costs of a garbled circuit of combined operations (exceeding the threshold).

Our method to incorporate a correction in the objective function is to add different (decreasing) costs for a respective operation x_i, depending on whether the previous operations $i' < i$ have been computed using garbled circuits ($z_{i'} = 1$). In order to limit the number of additional variables in the integer program, we consider at most $k = 20$ previous operations. Let $a_j(x_i)$ ($a_0(x_i) > \cdots > a_k(x_i)$) be the cost of an operation x_i if it and the previous j ($0 \leq j \leq k$) consecutive operations are executed as garbled circuits. We then introduce new variables $z'_{i,j}$ and replace each term $a(x_i)z'_i$ of operation i in the objective function by

$$a_0(x_i)z'_{i,0} + a_1(x_i)z'_{i,1} + \cdots + a_k(x_i)z'_{i,k}.$$

We add a constraint to allow only one new variable $z'_{i,j}$ per operation to be set to 1 such that only its cost is added

$$z'_{i,0} + \cdots + z'_{i,k} - z'_i = 0.$$

We then add constraints for previous operations that are executed as garbled circuits in order to select the correct (minimal) j'th cost $a_j(x_i)$

$$\begin{gathered}z'_{i,j} - z'_{i-0} \leq 0\\ \cdots\\ z'_{i,j} - z'_{i-j} \leq 0.\end{gathered}$$

The following constraints implement the conditions for the conversions based on the operator mapping function op. For each operation (element) $x_i \in \mathbb{X}$ and each of its operands $x_j \in op(x_i)$ we add the following constraint that determines whether x_j needs to be converted from garbled circuits to homomorphic encryption

$$z'_i - z'_j - z''_j \leq 0,$$

i.e., if z'_i is set (x_i is to be computed using garbled circuits), but z'_j is not set (x_j was computed using homomorphic encryption), then z''_j must be set (x_j must be converted).

Similarly, for each operation $x_i \in \mathbb{X}$ and each of its operands $x_j \in op(x_i)$ we add the following constraint that determines whether x_j needs to be converted from homomorphic encryption to garbled circuits

$$-z'_i + z'_j - z'''_j \leq 0,$$

i.e., if z'_i is not set (x_i is to be computed using homomorphic encryption) and z'_j is set (x_j was computed using garbled circuits), then z'''_j must be set (x_j must be converted).

Let $n = |\mathbb{X}|$ be the number of operations. Then, our integer program has $kn + 4n$ variables and at most $\frac{k(k-1)n}{2} + 5n$ constraints.

5.2 Heuristic

Integer programming is NP-complete and can become very slow for large instances. We therefore also implement a heuristic using a greedy algorithm. We start with all operations executed as garbled circuits. Then we consecutively scan each operation in a loop. If the overall cost decreases when converting this operation to homomorphic encryption we do so. We repeat until no more operations are converted.

The heuristic algorithm is shown in Algorithm 1. We use the same variables z'_i as above in §5.1 for each operation representing its assignment to either protocol type. We infer the variables z''_i and z'''_i using a helper routine and implement the remainder of the cost function in COST also as described above in §5.1. Initially we set all z'_i to 1 for garbled circuits (line 1). The algorithm has worst-case complexity $\mathcal{O}(n^2)$, since the inner loop (lines 6 - 17) is executed at most n times (at least one operation must be converted per iteration of the outer loop).

Algorithm 1 Cost-Driven Heuristic

```
Require: Cost function $\textsc{cost}(\cdot)$
Ensure: Partitioning $\boldsymbol{z}'$ of the operations into protocols
 1: $\boldsymbol{z}' \leftarrow \mathbf{1}$
 2: $cost \leftarrow \textsc{cost}(\boldsymbol{z}')$
 3: $flag \leftarrow 1$
 4: while $flag = 1$ do
 5:     $flag \leftarrow 0$
 6:     for $0 \leq i < n$ do
 7:         if $z'_i = 1$ then
 8:             $z'_i \leftarrow 0$
 9:             $c \leftarrow \textsc{cost}(\boldsymbol{z}')$
10:             if $c < cost$ then
11:                 $flag \leftarrow 1$
12:                 $cost \leftarrow c$
13:             else
14:                 $z'_i \leftarrow 1$
15:             end if
16:         end if
17:     end for
18: end while
```

6 Use Cases

In order to validate the complexity of manual partitioning and the cost advantage of our algorithmic approach, we consider three use cases for secure computation from the literature: joint economic-lot-size (§6.1), biometric identification (§6.2), and data mining (§6.3). Afterwards, we evaluate their performance in §6.4.

6.1 Secure Joint Economic Lot-Size

The secure joint economic lot-size problem describes a two-party scenario between a vendor and a buyer of a product. Both try to align the process of production, shipping, and warehousing according to an overall buyer's demand. Specifically, they try to agree on a joint lot-size q for production and shipping.

We call the demand known to both parties d, vendor's setup cost f_A, vendor's capacity c, supplier's ordering cost f_B, vendor's holding cost h_A, and supplier's holding cost h_B. Using the formula of [2] we can compute q as

$$q^2 = \frac{2 \cdot d \cdot f_A + 2 \cdot d \cdot f_B}{d \cdot \frac{h_A}{c} + h_B}.$$

The inputs to this calculation are sensitive (such as costs and capacities), since they disclose information about the cost calculation and influence future price negotiations if revealed. As described in [50] and can be seen above, the confidentiality-preserving computation of q can be reduced to secure division[1,32,9,10].

Secure division is also relevant for many other real world secure computations, e.g., k-means clustering [9]. As our use case we consider two division algorithms: the Newton-Raphson algorithm described in [1,54] and the long division algorithm described in [50]. That is, we compute for 32 bit inputs x and y held as shares x_A, y_A and x_B, y_B by the respective parties (cf. §3.2)

$$f(x_A, y_A, x_B, y_B) = \left\lfloor \frac{x_A + x_B}{y_A + y_B} \right\rfloor.$$

The Newton-Raphson algorithm has 302 operations in the intermediate language and the long division algorithms has 383 operations.

6.2 Biometric Identification

Comparing and matching biometric data is a highly privacy-sensitive task in systems that are widely used in law enforcement, including fingerprint-, iris-, and face-recognition systems[14,53,48,25,5,3]. The identification is based on comparing the submitted biometric information to values in a database, determining the closest match with respect to some metric (e.g., Euclidean distance).

As our use case we consider an algorithm for biometric identification, computing the distances using Euclidean distance as metric which is commonly used for fingerprints and faces. That is, we compute

$$min\left(\sum_{i=1}^{M}(S_{1,i} - C_i)^2, \cdots, \sum_{i=1}^{M}(S_{N,i} - C_i)^2\right)$$

for $N = 5$ vectors of $M = 4$ elements $S_{i,j}$ in the server database and a client vector C_i of M elements, for elements of 32 bit. The algorithm has 80 operations in the intermediate language.

6.3 Data Mining

Data mining aims to extract knowledge from databases, connecting the worlds of databases, artificial intelligence, and statistics. Various data mining algorithm for different purposes have been proposed in the literature. One particular purpose is that of structuring data sets in order to provide decision mechanisms that can be used for classification. A well known algorithm for decision tree learning is the ID3 algorithm described in [52]. A privacy-preserving classification variant of ID3, described in [35] as one of the first privacy-preserving data mining algorithms, enables new applications where multiple private databases can be used to act as training set (e.g., medical databases). The authors of [35] use entropy to compute the best attributes, with the privacy-preserving computation of the natural logarithm as the basis operation.

As our use case we consider an algorithm to compute the natural logarithm. To the best of our knowledge, this is the first implementation of this privacy-preserving data mining algorithm. That is, we compute the natural logarithm of a 32 bit input $x = 2^n(1 + \epsilon)$ held as shares x_A and x_B by the respective parties where 2^n is the power of 2 which is closest to x and $-1/2 \leq \epsilon < 1/2$. The natural logarithm is approximated with a Taylor series with $k = 10$ iterations:

$$\ln(x) = \ln(2^n(1 + \epsilon)) = n \ln 2 + \epsilon - \frac{\epsilon^2}{2} + \cdots \frac{\epsilon^k}{k}.$$

The algorithm has 270 operations in the intermediate language.

6.4 Evaluation

In the following we evaluate our partitioning algorithms of §5 on the three use cases introduced in §6.1, §6.2, and §6.3. Using these results we compare the performance of mixed protocols to garbled circuit protocols,

Security	Partitioning	LD		NR		ED		LOG	
		LAN	WAN	LAN	WAN	LAN	WAN	LAN	WAN
short-term	HE-only	81.6	104.7	93.6	119.5	14.8	18.4	72.6	95.7
	GC-only	2.0	5.6	12.3	82.2	6.1	16.6	1.6	4.0
	Heuristic	2.0	5.5	12.3	59.5	6.0	11.4	1.5	3.3
	IP	2.0	5.5	12.3	59.5	6.0	11.4	1.4	3.1
mid-term	HE-only	588.6	611.7	675.0	700.9	106.8	110.4	523.3	546.4
	GC-only	2.1	7.8	12.1	115.0	5.9	23.1	1.6	5.4
	Heuristic	2.0	7.7	12.1	115.0	5.8	22.8	1.4	4.6
	IP	2.0	7.6	12.1	115.0	5.8	22.7	1.4	4.4
long-term	HE-only	1,974.2	1,997.3	2,264.9	2,290.9	359.5	363.1	1,749.2	1,772.4
	GC-only	2.1	8.8	12.1	131.4	5.9	26.3	1.6	6.2
	Heuristic	2.0	8.8	12.1	131.4	5.8	25.9	1.4	5.2
	IP	2.0	8.6	12.1	131.4	5.8	25.9	1.4	4.9

Table 1. Runtime forecasts in [seconds] for long division (LD), Newton-Raphson (NR), Euclidean distance (ED), and natural logarithm (LOG) on 32 bit inputs.

Security	Partitioning	LD	NR	ED	LOG
short-term	HE-only	776	852	96	892
	GC-only	5,057	76,668	15,147	3,384
	Heuristic	5,012	76,668	14,877	3,361
	IP	4,938	76,668	14,945	2,648
mid-term	HE-only	1,552	1,704	192	1,784
	GC-only	7,080	107,336	21,192	4,737
	Heuristic	7,017	107,336	20,814	4,706
	IP	6,914	107,336	20,908	3,692
long-term	HE-only	2,328	2,556	288	2,676
	GC-only	8,092	122,669	24,214	5,414
	Heuristic	8,020	122,669	23,782	5,378
	IP	7,901	122,669	23,890	4,214

Table 2. Communication forecasts in [kb] for long division (LD), Newton-Raphson (NR), Euclidean distance (ED), and natural logarithm (LOG) on 32 bit inputs.

the optimization of the heuristic to that of integer programming, and the automatic optimal partitioning to the manual partitioning approach.

As execution environment of the secure computation protocols we consider a LAN environment (bandwidth $b = 100$ Mbit/s, latency $t_{LAT} = 1$ μs) and a WAN environment (bandwidth $b = 1$ Mbit/s, latency $t_{LAT} = 100$ ms). The performance of local operations has been measured on servers with four AMD Opteron 885 dual-core 64-bit CPUs and 16 GB RAM using a single-threaded implementation (cf. §C for details). We use Java Version 6 and instantiate security parameters according to NIST recommendations [47] (cf. §B for details).

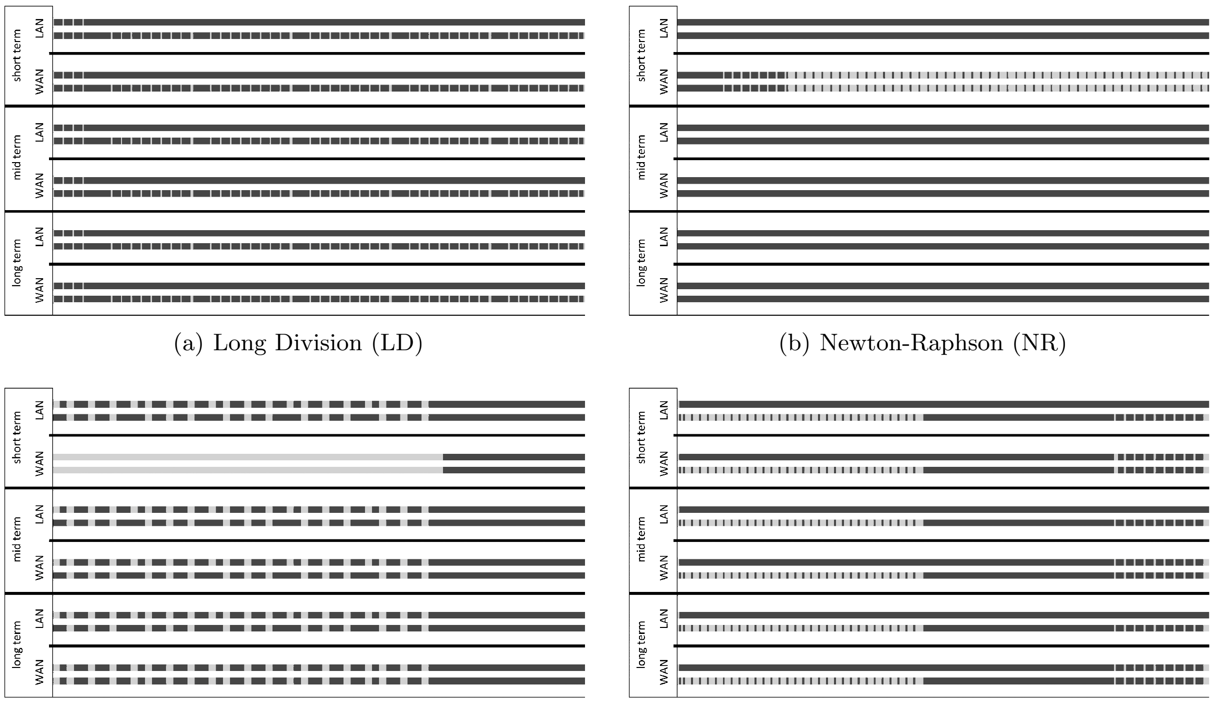
In a brief experimental study we confirmed the accuracy of the performance model described in §4. We executed all four use cases in the LAN/WAN setting with short-term security (80 bit) using the mixed partitioning. Our forecasts were always within the same error bound as reported in [54].

Tab. 1 summarizes the runtime forecasts for algorithms long division, Newton-Raphson, Euclidean distance, and natural logarithm. The table consists of the respective results in seconds for partitions that are computed entirely using homomorphic encryption (HE-only) or garbled circuits (GC-only), and for mixed partitions that were found by our heuristic and by integer programming (IP).

Mixed versus Non-mixed Protocols. The results in Tab. 1 show that for our use cases, mixed protocols can reduce runtimes below those of single protocols. For pure homomorphic encryption and garbled circuits we draw two conclusions.

First, in all use cases and settings the homomorphic encryption protocols result in highest runtimes. In particular for growing key lengths of mid- and long-term security settings, homomorphic encryption is slower than garbled circuits by orders of magnitudes.

Second, garbled circuit protocols are sometimes competitive, but may be improved by mixed protocols. In 16 out of 24 experimental settings, garbled circuits have runtimes close to the best results (not more than 5% deviation). In four cases the garbled circuit protocol results in the best performance. In all experimental settings, both partitioning mechanisms for computing optimal mixed protocols result in the best performance,



(a) Long Division (LD)

(b) Newton-Raphson (NR)

(c) Euclidean Distance (ED)

(d) Natural Logarithm (LOG)

Fig. 1. Partitioning of algorithms for 32 bit inputs. Operations computed using GC are depicted in dark-gray, those computed using HE in gray. The upper and lower bar depict the partitioning found by the heuristic and integer program, respectively.

including the previously mentioned four pure garbled circuit cases. In 8 of 24 settings, the mixed protocols result in an average of 20% less runtime. The largest improvement is 31% lower runtime compared to the protocol entirely implemented as garbled circuit (Euclidean distance, short-term security, WAN).

We infer that network conditions are essential in the context of performance measurements. For LAN settings, mixed protocols obtain on average an improvement over the garbled circuit protocol of 4%. For WAN settings, however, the improvement is significantly higher, namely 11%.

Tab. 2 depicts the communication complexities of the protocols in kilobytes. Clearly, non-mixed protocols yield either most (GC-only) or least (HE-only) communication traffic. From the perspective of communication and related costs (e.g., fees charged in mobile networks), HE-only as well as mixed protocols clearly have an advantage over GC-only protocols. The reason are the corresponding operators and sub-protocols of HE sub-protocols that, compared to GC-only protocols, have to transmit only a low amount of data. Additional communication savings can be obtained by packing these data. A good example is the use-case of calculating the Euclidean distance. Considering the amount of data to transmit, in Tab. 2 we see a difference of magnitudes between GC-only and HE-only and a reasonable difference to the mixed protocol. Regarding the example, a joint view on Tab. 1 and Tab. 2 shows how a mixed protocol (by both, Heuristic and IP) can significantly reduce runtime as well as network traffic.

Heuristic versus Integer Programming. Both optimization approaches result in mixed protocols that perform, in almost half of all experimental settings, noticeably better than pure protocols. As seen from the results in Tab. 1, the heuristic based partitioning results are close to those of integer programming (deviating not more than 2.7% on average, at maximum 7.6%). While the heuristic only requires seconds to compute the partitioning per use case and setting, the integer program requires several hours using the LP solver SoPlex[3] on the aforementioned server hardware. While the performance of the mixed protocols found by the two partitioning algorithms is similar, the resulting partitionings differ in several aspects (see Tab. 4 and Tab. 5 in §D for details). The heuristic, in comparison to the integer program, tends to reduce the number of blocks. A block is a sequence of consecutive operations with the same protocol type. For long division and natural logarithm, over all settings, the ratio between number of blocks and number of operations is less than 0.025, while it is more than 0.279 (i.e., larger by a factor of 10) for the integer program. On the contrary,

[3] version 1.6.0, available at http://soplex.zib.de/

results for Netwton-Raphson and Euclidean distance show that both partitioning algorithms may result in similarly high (0.5) or low (0.003) ratios.

Manual versus Automated Partitioning. Using an analysis of the partitions found by our algorithms – independent of heuristic or integer program – we argue that it is complicated to find the same partition manually. Fig. 1 shows how the optimization approaches partitioned the use cases in the various settings. Operations computed using garbled circuits are depicted in dark-gray, those computed using homomorphic encryption in gray.

Fig. 1 shows that the mixed protocols are heavily fragmented in order to achieve the optimal performance (details are given in Tab. 5 in §D). We obtain a wide spectrum of fragmentations. For Euclidean distance we have 40 blocks (of at most two operations per block) within only 80 operations in total. Similarly, for Newton-Raphson we obtain 113 blocks (of 1 to 26 operations per block) within 302 operations. Regarding partitions with at least two blocks, we obtain the largest block for natural logarithm (of 221 operations) within 270 operations.

Although there seem to be patterns in some areas of the diagrams, it is difficult to infer a general conclusion that can be used to manually derive a partitioning with similar performance. Fig. 1 shows that for some sub-sequences partitions are constant (within the same network setting but for changing security levels, e.g., long division). Others change within the same network setting for changing security levels (e.g., Euclidean distance and Newton-Raphson). In only 3 out of 12 cases there is no change in the partitioning across different network settings.

Even unrolled operation blocks that are identical on the operation level, result in different partitionings within the same setting and use case. One such example is the natural logarithm; operations that are part of the main loop last from the middle of the algorithm until the (third) last operation.

7 Conclusions

In this paper we have presented algorithms to automatically select a protocol – garbled circuits or homomorphic encryption – in secure two-party computation. Based on a performance model our algorithms minimize the costs of a mixed protocol. We present an evaluation based on three use cases from the literature: secure joint economic lot-size, biometric identification, and data mining.

Our results support that mixed protocols perform better than pure garbled circuit implementations. In 8 out of 24 experiments we achieve a performance gain of 20% on average. We conclude that the option to mix protocols improves performance of secure two-party computation.

Our results also support that our heuristic is close to the optimization algorithm based on integer programming. In all experiments our heuristic achieved a performance within 2.7% of the optimum on average. Nevertheless, the heuristic runs within seconds whereas the integer program requires hours. We conclude that it is practically feasible to automatically do the (near-optimal) selection.

Furthermore, our detailed analysis of the experiments also revealed that there is no discernible pattern of the selection. A programmer cannot rely on simple hints in order to perform the selection manually. We therefore conclude that the protocol selection problem is too complicated to be solved manually by the programmer and needs to be solved automatically, e.g., by a compiler.

Further work is to extend the protocols and selection to stronger security models, such as the malicious model.

Acknowledgements. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 609611 (PRACTICE). Thomas Schneider was supported by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE and by the Hessian LOEWE excellence initiative within CASED.

References

1. M. Atallah, M. Bykova, J. Li, K. Frikken, M. Topkara. Private Collaborative Forecasting and Benchmarking. *ACM Privacy in the Electronic Society (WPES)*, 2004.
2. A. Banerjee. A Joint Economic-Lot-Size Model For Purchaser and Vendor. *Decision Sciences 17(3)*, 1986.
3. M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Donida Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, A. Piva. Privacy-Preserving Fingercode Authentication. *ACM Multimedia and Security (MMSEC)*, 2010.
4. A. Ben-David, N. Nisan, B. Pinkas. FairplayMP: A System for Secure Multi-Party Computation. *ACM Computer and Communications Security (CCS)*, 2008.

5. M. Blanton, P. Gasti. Secure and Efficient Protocols for Iris and Fingerprint Identification. *European Symposium on Research in Computer Security (ESORICS)*, 2011.
6. D. Bogdanov, S. Laur, J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. *European Symposium on Research in Computer Security (ESORICS)*, 2008.
7. D. Bogdanov, R. Talviste, J. Willemson. Deploying Secure Multi-Party Computation for Financial Data Analysis. *Financial Cryptography (FC)*, 2012.
8. P. Bogetoft, D.L. Christensen, I. Damgård, M. Geisler, T.P. Jakobsen, M. Krøigaard, J.D. Nielsen, J.B. Nielsen, K. Nielsen, J. Pagter, M.I. Schwartzbach, T. Toft. Secure Multiparty Computation Goes Live. *Financial Cryptography (FC)*, 2009.
9. P. Bunn, R. Ostrovsky. Secure Two-Party k-Means Clustering. *ACM Computer and Communications Security (CCS)*, 2007.
10. O. Catrina, A. Saxena. Secure Computation with Fixed-Point Numbers. *Financial Cryptography (FC)*, 2010.
11. S.G. Choi, K.-W. Hwang, J. Katz, T. Malkin, D. Rubenstein. Secure Multi-Party Computation of Boolean Circuits with Applications to Privacy in On-Line Marketplaces. *The Cryptographers' Track at the RSA Conference (CT-RSA)*, 2012.
12. I. Damgård, M. Geisler, M. Krøigaard, J.B. Nielsen. Asynchronous Multiparty Computation: Theory and Implementation. *Public Key Cryptography (PKC)*, 2009.
13. I. Damgård, R. Thorbek. Efficient Conversion of Secret-Shared Values Between Different Fields. *Cryptology ePrint Archive: Report 2008/221*, 2008.
14. Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, T. Toft. Privacy-Preserving Face Recognition. *Privacy Enhancing Technologies Symposium (PETS)*, 2009.
15. C. Gentry. Fully Homomorphic Encryption using Ideal Lattices. *ACM Symposium on Theory of Computing (STOC)*, 2009.
16. C. Gentry, S. Halevi. Implementing Gentry's Fully-Homomorphic Encryption Scheme. *Advances in Cryptology – EUROCRYPT*, 2011.
17. B. Goethals, S. Laur, H. Lipmaa, T. Mielikäinen. On Private Scalar Product Computation for Privacy-Preserving Data Mining. *International Conference on Information Security and Cryptology (ICISC)*, 2004.
18. O. Goldreich. Foundations of Cryptography: Volume 2 – Basic Applications. Cambridge Univ. Press, 2004.
19. O. Goldreich, S. Micali, A. Wigderson. How to Play Any Mental Game. *ACM Symposium on Theory of Computing (STOC)*, 1987.
20. W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. *ACM Computer and Communications Security (CCS)*, 2010.
21. A. Holzer, M. Franz, S. Katzenbeisser, H. Veith. Secure Two-Party Computation in ANSI C. *ACM Computer and Communications Security (CCS)*, 2012.
22. Y. Huang, P. Chapman, D. Evans. Privacy-Preserving Applications on Smartphones. *USENIX Hot Topics in Security (HotSec)*, 2011.
23. Y. Huang, D. Evans, J. Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? *Network and Distributed System Security (NDSS)*, 2012.
24. Y. Huang, D. Evans, J. Katz, L. Malka. Faster Secure Two-Party Computation Using Garbled Circuits. *USENIX Security Symposium*, 2011.
25. Y. Huang, L. Malka, D. Evans, J. Katz. Efficient Privacy-Preserving Biometric Identification. *Network and Distributed System Security (NDSS)*, 2011.
26. Y. Ishai, J. Kilian, K. Nissim, E. Petrank. Extending Oblivious Transfers Efficiently. *Advances in Cryptology – CRYPTO*, 2003.
27. F. Kerschbaum. Practical Privacy-Preserving Benchmarking. *IFIP International Information Security Conference (SEC)*, 2008.
28. F. Kerschbaum. Automatically Optimizing Secure Computation. *ACM Computer and Communications Security (CCS)*, 2011.
29. F. Kerschbaum, A. Schröpfer, A. Zilli, R. Pibernik, O. Catrina, S. de Hoogh, B. Schoenmakers, S. Cimato, E. Damiani. Secure Collaborative Supply Chain Management. *IEEE Computer 44 (9)*, 2011.
30. F. Kerschbaum, T. Schneider, A. Schröpfer. Automatic Protocol Selection in Secure Two-Party Computations. *Network and Distributed System Security (NDSS)*, 2013.
31. F. Kerschbaum, T. Schneider, A. Schröpfer. Automatic Protocol Selection in Secure Two-Party Computations. *Applied Cryptography and Network Security (ACNS)*, 2014.
32. E. Kiltz, G. Leander, J. Malone-Lee. Secure Computation of the Mean and Related Statistics. *Theory of Cryptography Conference (TCC)*, 2005.
33. V. Kolesnikov, A.-R. Sadeghi, T. Schneider. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. *Cryptology And Network Security (CANS)*, 2009.
34. V. Kolesnikov, T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. *International Colloquium on Automata, Languages and Programming (ICALP)*, 2008.
35. Y. Lindell, B. Pinkas. Privacy Preserving Data Mining. *Journal of Cryptology 15(3)*, 2002.
36. Y. Lindell, B. Pinkas. A Proof of Yao's Protocol for Secure Two-Party Computation. *Journal of Cryptology 22(2)*, 2009.
37. P.D. MacKenzie, A. Oprea, M.K. Reiter. Automatic Generation of Two-Party Computations. *ACM Computer and Communications Security (CCS)*, 2003.

38. L. Malka. VMCrypt - Modular Software Architecture for Scalable Secure Computation. *ACM Computer and Communications Security (CCS)*, 2011.
39. D. Malkhi, N. Nisan, B. Pinkas, Y. Sella. Fairplay - A Secure Two-party Computation System. *USENIX Security Symposium*, 2004.
40. T. M. Mitchell, *Machine Learning.* McGraw-Hill, 1997.
41. B. Mood, L. Letaw, K. Butler. Memory-Efficient Garbled Circuit Generation for Mobile Devices. *Financial Cryptography (FC)*, 2012.
42. M. Naor, B. Pinkas. Efficient Oblivious Transfer Protocols. *Symposium on Data Structures and Algorithms (SODA)*, 2001.
43. M. Naor, B. Pinkas, R. Sumner. Privacy Preserving Auctions and Mechanism Design. *ACM Conference on Electronic Commerce (EC)*, 1999.
44. J. B. Nielsen, P. S. Nordholt, C. Orlandi, S. S. Burra. A New Approach to Practical Active-Secure Two-Party Computation. *Advances in Cryptology – CRYPTO*, 2012.
45. V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, D. Boneh. Privacy-preserving Matrix Factorization. *ACM Computer and Communications Security (CCS)*, 2013.
46. V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, N. Taft. Privacy-Preserving Ridge Regression on Hundreds of Millions of Records. *IEEE Security and Privacy (S&P)*, 2013.
47. NIST. Recommendation for Key Management. Special Publication 800-57 Part 1 Rev. 3, 07/2012.
48. M. Osadchy, B. Pinkas, A. Jarrous, B. Moskovich. Scifi - A System for Secure Face Identification. *IEEE Security and Privacy (S&P)*, 2010.
49. P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Advances in Cryptology – EUROCRYPT*, 1999.
50. R. Pibernik, Y. Zhang, F. Kerschbaum, A. Schröpfer. Secure Collaborative Supply Chain Planning and Inverse Optimization - The JELS Model. *European Journal of Operational Research (EJOR) 208(1)*, 2011.
51. B. Pinkas, T. Schneider, N.P. Smart, S.C. Williams. Secure Two-party Computation is Practical. *Advances in Cryptology - ASIACRYPT*, 2009.
52. J.R. Quinlan. Induction of Decision Trees. *Machine Learning 1(1)*, 1986.
53. A.-R. Sadeghi, T. Schneider, I. Wehrenberg. Efficient Privacy-Preserving Face Recognition. *International Conference on Information Security and Cryptology (ICISC)*, 2009.
54. A. Schröpfer, F. Kerschbaum. Forecasting Run-Times of Secure Two-Party Computation. *Int. Conference on Quantitative Evaluation of Systems (QEST)*, 2011.
55. A. Schröpfer, F. Kerschbaum, G. Müller. L1 - An Intermediate Language for Mixed-Protocol Secure Computation. *IEEE Computer Software and Applications Conference (COMPSAC)*, 2011.
56. A.C. Yao. How to Generate and Exchange Secrets. *IEEE Foundations of Computer Science (FOCS)*, 1986.

A Updated Performance Model for Garbled Circuits

We extend and adapt the performance model of [54] (which was based on Yao's garbled circuit protocol as implemented in Fairplay [39]) to reflect state-of-the-art optimizations of Yao's garbled circuits protocol as implemented in FastGC [24] and VMCrypt [38]: For garbled circuits we use free XORs of [34], garbled row reduction of [43,51], and pipelining of [24]. For oblivious transfer (OT) we use the OT extension protocol of [26] together with the OT protocol of [42, Sect. 3] for the base OTs. We denote with α^A and α^B the number of input bits of Alice and Bob, and with β^A and β^B their number of private output bits.

A.1 Optimized GC Construction

Let k_{GC} be the length of symmetric keys used in the garbled circuit construction. Using the free XOR technique of [34], a random key of length k_{GC} needs to be chosen for the key difference and each input bit of A and B. Using the garbled row reduction technique of [43,51], the random keys for the outputs of the binary gates are determined given the random keys of the inputs and no longer need to be chosen at random. Let $t^A_{RND}(n)$ be the time to choose n random bits by Alice. Then, the overall time to choose the random keys is reduced to approximately $t_{rand} = (1 + \alpha^A + \alpha^B)t^A_{RND}(k_{GC})$.

Due to the free XOR technique of [34] that requires only negligible computation and no communication for XOR gates we set n_g to the number of non-XOR gates (in the original model of [54] this was the total number of gates). For the basic operations we use the circuits of [33] that are optimized to have a small number of non-XOR gates: $n_g(\oplus) = n_g(\ominus) = n_g(\leq) = l$, where l is the bit length of the operands. Similarly, we have $n_g(\odot_e) = (2l^2 - l + 4)e$ and $n_g(\odot_c) = l(d_H(c) - 1)$, where $d_H(c)$ is the Hamming weight of c.

The garbled row reduction technique of [43,51] results in only 3 encrypted table entries per non-XOR gate, i.e., approximately $3k_{GC}$ bits. Let $t_{msg}(s)$ denote the time required for transferring a message of size s bits, i.e., $t_{msg}(s) = s/r_{t_{LAT},b}(s)$ where $r_{t_{LAT},b}(n)$ is the transfer rate for sending n bits (depending on bandwidth b and latency t_{LAT} as used in [54]). Furthermore, let t^A_{OWH} and t^B_{OWH} denote the time for computing the one-way hash function used for symmetric encryption of a garbled circuit gate entry by Alice and Bob, respectively. [24] proposed to pipeline garbled circuits, i.e., for each gate the encrypted table entries are generated by Alice, sent directly to Bob, and evaluated by Bob. Hence, the total time for streaming, i.e., generating, transferring, and evaluating, the garbled circuit is $t_{GC} = n_g \max(4t^A_{OWH}(2k_{GC}), t_{msg}(3k_{GC}), t^B_{OWH}(2k_{GC}))$.

A.2 Optimized OTs

The OT extension technique of [26] allows to reduce a large number of 1-out-of-2 OTs to only σ 1-out-of-2 OTs (executed with exchanged roles of Alice and Bob), where σ is a statistical security parameter set to $\sigma = 80$ in our implementation. In addition to this, only a small number of invocations of a cryptographic hash function (modeled as random oracle) needs to be performed.

In our mixed protocols we need to run parallel OTs at several places, namely whenever we evaluate a garbled circuit to which Bob provides inputs. In particular, this is always the case when converting from homomorphic encryption to garbled circuits as described in §3.3. Let l_i be the number of parallel OTs that need to be performed in chunk i. W.l.o.g. we assume that the total number of OTs $\sum_i l_i > \sigma$, i.e., we always use the online version of the OT extension technique of [26] as described in detail in §A.4.

The costs for the setup phase are dominated by the costs for performing σ parallel OTs using the protocol of [42, Sect. 3] over an order q subgroup of $\mathbb{Z}^*_p$, e.g., $|p| = 1{,}024, |q| = 128$ as used in the implementation of [24]. Let t_{LAT} be the latency of the network connection and let t^X_{POW}, t^X_{MUL}, and t^X_{INV} denote the times for modular arithmetic operations of exponentiation, multiplication and inversion, respectively, by party X (Alice A or Bob B). The total time required to run this protocol can be estimated by $t^{setup}_{OT} = \sigma(3t^B_{POW}(|p|,|q|) + t^B_{INV}(|p|) + t^B_{MUL}(|p|) + 2t^B_{OWH}(|p|+\sigma) + 2t^A_{POW}(|p|,|q|) + t^A_{INV}(|p|) + t^A_{MUL}(|p|) + t^A_{OWH}(|p|+\sigma)) + 3t_{LAT} + t_{msg}(|p|) + t_{msg}(\sigma|p|) + t_{msg}(\sigma(|p| + 2k_{GC}))$. We emphasize that this cost accounts only once per protocol.

The costs for running the i-th chunk of l_i parallel OTs can be approximated by $t_{OT}(l_i) = 2\sigma t^B_{OWH}(k_{GC})\frac{l_i}{2k_{GC}} + \sigma t^A_{OWH}(k_{GC})\frac{l_i}{2k_{GC}} + 2l_i t^A_{OWH}(\sigma) + l_i t^B_{OWH}(\sigma) + 2t_{LAT} + t_{msg}(2\sigma l_i) + t_{msg}(2l_i k_{GC})$ (cf. §A.4 for details).

A.3 Overall

The time (cost) for the entire garbled circuit protocol as implemented in [24] is the sum of the times for

- choosing random wire labels t_{rand},
- sending the wire labels for A's inputs $t_{msg}(\alpha^A k_{GC})$,
- sending the wire labels for B's inputs via OT $t_{OT}(\alpha^B)$,
- streaming the garbled circuit t_{GC},
- sending A's encrypted outputs $t_{msg}(\beta^A k_{GC})$, and
- sending the output decryption information for B's outputs $t_{msg}(2\beta^B k_{GC})$.

A.4 Online Efficient OT Extension

The online version of the efficient OT extension protocol of [26] is depicted in Fig. 2. $\mathbf{T}$ is a $l_i \times \sigma$ bit matrix, where $\mathbf{t}^j$ denotes its j-th column and $\mathbf{t}_j$ its j-th row. The random oracle H can be instantiated with a cryptographic hash function whose output length is twice as large as the symmetric security parameter used in the garbled circuit protocol, e.g., H=SHA-256 for $k = k_{GC} = 128$. For simplicity of presentation we assume in Fig. 2 that the chunk length l_i is exactly the output length of the hash function.

B Security Parameters

We set the security parameters according to recommendations by NIST [47]:

- short-term security (recommended until 2010): size of RSA modulus in the homomorphic cryptosystem $k_{HE} = |p| = 1{,}024$, garbled circuit key-length $k_{GC} = 80$ and $|q| = 160$ (using SHA-1 as OWH function)
- mid-term security (recommended 2011-2030): size of RSA modulus in the homomorphic cryptosystem $k_{HE} = |p| = 2{,}048$, garbled circuit key-length $k_{GC} = 112$ and $|q| = 224$ (using SHA-224 as OWH function)
- long-term security (recommended $>$ 2030): size of RSA modulus in the homomorphic cryptosystem $k_{HE} = |p| = 3{,}072$, garbled circuit key-length $k_{GC} = 128$ and $|q| = 256$ (using SHA-256 as OWH function).

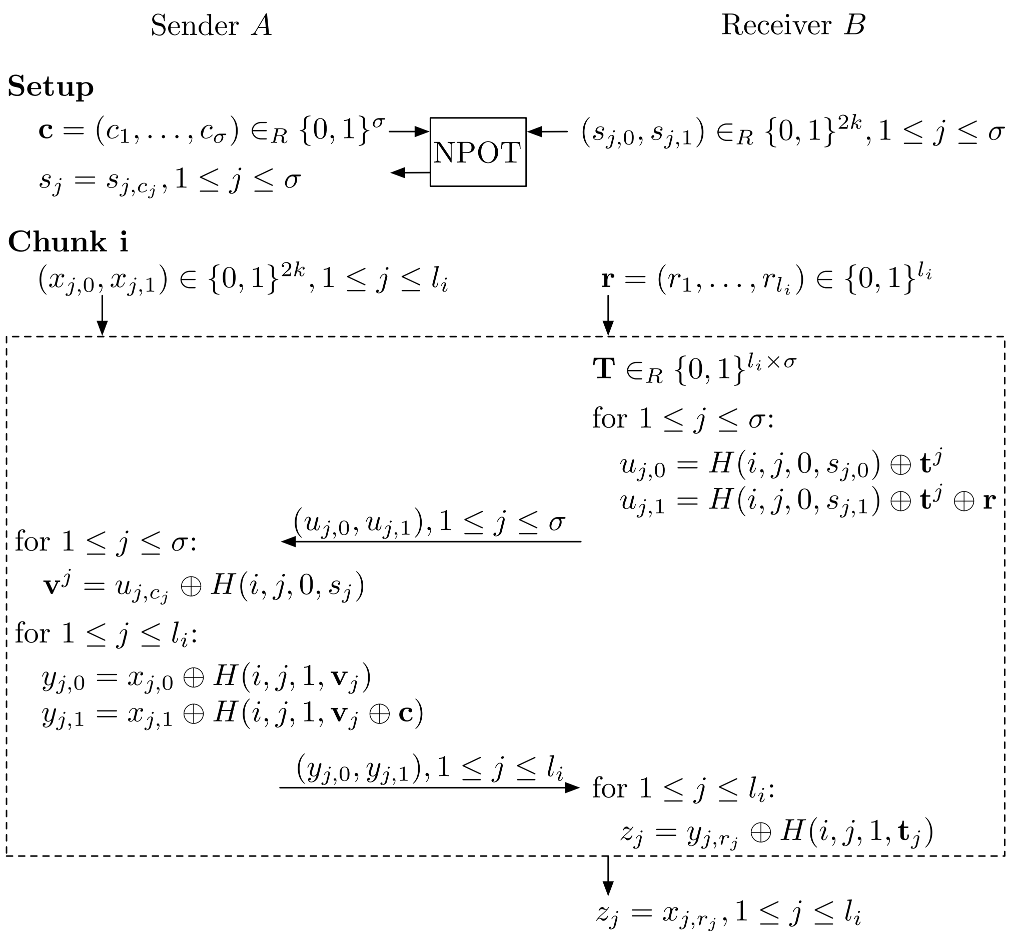

Sender A
Receiver B
Setup
$\mathbf{c} = (c_1, \ldots, c_\sigma) \in_R \{0,1\}^\sigma$
NPOT
$(s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2k}, 1 \le j \le \sigma$
$s_j = s_{j,c_j}, 1 \le j \le \sigma$
Chunk i
$(x_{j,0}, x_{j,1}) \in \{0,1\}^{2k}, 1 \le j \le l_i$
$\mathbf{r} = (r_1, \ldots, r_{l_i}) \in \{0,1\}^{l_i}$
$\mathbf{T} \in_R \{0,1\}^{l_i \times \sigma}$
for $1 \le j \le \sigma$:
$u_{j,0} = H(i, j, 0, s_{j,0}) \oplus \mathbf{t}^j$
$u_{j,1} = H(i, j, 0, s_{j,1}) \oplus \mathbf{t}^j \oplus \mathbf{r}$
$(u_{j,0}, u_{j,1}), 1 \le j \le \sigma$
for $1 \le j \le \sigma$:
$\mathbf{v}^j = u_{j,c_j} \oplus H(i, j, 0, s_j)$
for $1 \le j \le l_i$:
$y_{j,0} = x_{j,0} \oplus H(i, j, 1, \mathbf{v}_j)$
$y_{j,1} = x_{j,1} \oplus H(i, j, 1, \mathbf{v}_j \oplus \mathbf{c})$
$(y_{j,0}, y_{j,1}), 1 \le j \le l_i$
for $1 \le j \le l_i$:
$z_j = y_{j,r_j} \oplus H(i, j, 1, \mathbf{t}_j)$
$z_j = x_{j,r_j}, 1 \le j \le l_i$

Fig. 2. Online Efficient OT Extension of [26].

Security	Parameter	Description	Time
short	$t_{OWH}(80)_{10^3}$	hash function in a batch of 10^3 numbers of 80 bits	0.057
term	$t_{OWH}(80)_{10^4}$	hash function in a batch of 10^4 numbers of 80 bits	0.024
	$t_{OWH}(80)_{10^5}$	hash function in a batch of 10^5 numbers of 80 bits	0.007
	$t_{ADD}(1{,}024)$	add two 1,024 bit numbers	0.009
	$t_{MUL}(1{,}024)$	multiply two 1,024 bit numbers	0.404
	$t_{POW}(1{,}024)$	modular exponentiation of a 1,024 bit number	13.222
	$t_{ENC}(1{,}024)$	homomorphic encryption with 1,024 bit key	95.331
	$t_{DEC}(1{,}024)$	homomorphic decryption with 1,024 bit key	47.779
	$t_{2POW}(1{,}024)$	modular exponentiation of a 1,024 bit number with squared modulus	45.825
	$t_{POW}(160)$	modular exponentiation of 160 bit number	1.116
	$t_{rand}(80)_{10^3}$	select in a batch of 10^3 a set of 80 random bits	0.031
	$t_{rand}(80)_{10^4}$	select in a batch of 10^4 a set of 80 random bits	0.213
	$t_{rand}(80)_{10^5}$	select in a batch of 10^5 a set of 80 random bits	0.626
	$t_{POW}(1{,}024; 160)$	modular exponentiation of 1,024 bit base with 160 bit exponent	2.988
	$t_{INV}(1{,}024)$	modular inversion of 1,024 bit number	0.546
mid	$t_{OWH}(112)_{10^3}$	hash function in a batch of 10^3 numbers of 112 bits	0.058
term	$t_{OWH}(112)_{10^4}$	hash function in a batch of 10^4 numbers of 112 bits	0.026
	$t_{OWH}(112)_{10^5}$	hash function in a batch of 10^5 numbers of 112 bits	0.007
	$t_{ADD}(2{,}048)$	add two 2,048 bit numbers	0.015
	$t_{MUL}(2{,}048)$	multiply two 2,048 bit numbers	0.185
	$t_{POW}(2{,}048)$	modular exponentiation of 2,048 bit number	82.892
	$t_{ENC}(2{,}048)$	homomorphic encryption with 2,048 bit key	705.142
	$t_{DEC}(2{,}048)$	homomorphic decryption with 2,048 bit key	318.545
	$t_{2POW}(2{,}048)$	modular exponentiation a 2,048 bit number with squared modulus	338.000
	$t_{POW}(224)$	modular exponentiation of two 224 bit numbers	1.168
	$t_{rand}(112)_{10^3}$	select in a batch of 10^3 a set of 112 random bits	0.032
	$t_{rand}(112)_{10^4}$	select in a batch of 10^4 a set of 112 random bits	0.223
	$t_{rand}(112)_{10^5}$	select in a batch of 10^5 a set of 112 random bits	0.655
	$t_{POW}(2{,}048; 224)$	modular exponentiation of 2,048 bit base with 224 bit exponent	3.729
	$t_{INV}(2{,}048)$	modular inversion of 2,048 bit number	0.764
long	$t_{OWH}(128)_{10^3}$	hash function in a batch of 10^3 numbers of 128 bits	0.060
term	$t_{OWH}(128)_{10^4}$	hash function in a batch of 10^4 numbers of 128 bits	0.027
	$t_{OWH}(128)_{10^5}$	hash function in a batch of 10^5 numbers of 128 bits	0.007
	$t_{ADD}(3{,}072)$	add two 3,072 bit numbers	0.039
	$t_{MUL}(3{,}072)$	multiply two 3,072 bit numbers	0.339
	$t_{POW}(3{,}072)$	modular exponentiation of 3,072 bit number	304.657
	$t_{ENC}(3{,}072)$	homomorphic encryption with 3,072 bit key	2,359,147
	$t_{DEC}(3{,}072)$	homomorphic decryption with 3,072 bit key	1,144,681
	$t_{2POW}(3{,}072)$	modular exponentiation of 3,072 bit number with squared modulus	1,142,168
	$t_{POW}(256)$	modular exponentiation of two 256 bit numbers	1.220
	$t_{rand}(128)_{10^3}$	select in a batch of 10^3 a set of 128 random bits	0.034
	$t_{rand}(128)_{10^4}$	select in a batch of 10^4 a set of 128 random bits	0.233
	$t_{rand}(128)_{10^5}$	select in a batch of 10^5 a set of 128 random bits	0.685
	$t_{POW}(3{,}072; 256)$	modular exponentiation of 3,072 bit base with 256 bit exponent	4.374
	$t_{INV}(3{,}072)$	modular inversion of 3,072 bit number	0.873

Table 3. Times in [ms] of local operations measured for party A at calibrating performance model of [54] on hardware introduced in §6.4 (note: hardware of party A and B is identical).

C Performance Model Calibration

In Tab. 3 we present the calibration results for setting up the performance model of [54]. The table shows the absolute consumed clock-time of the local operators. Those values are used to precisely calculate the run-times of the high-level (protocol) operators in the performance model as explained in detail in [54].

D Evaluation Results

D.1 Details on Manual versus Automated Partitioning

One could assume that there would be a rather intuitive relation between single operations in the intermediate language and both types of discussed protocols. Intuitively, for shared values (which we designed to be part of the homomorphic encryption model), operations can be assumed to be fast, if they are executed as local operations that do not use cryptographic algorithms (e.g., addition or multiplication by a constant). Similarly, garbled circuits could be supposed to perform faster than homomorphic encryption for comparing two secret values. Tab. 4 shows the number of operations performed using garbled circuits or homomorphic encryption in the mixed protocols found by our selection algorithms. These metrics show that the relations are rather complex. For Newton-Raphson and short-term security, both algorithms assign the majority of subtraction operations to homomorphic encryption (48 of 60), since these operations can be implemented locally without communication. In contrast, for long division in the same security setting, both algorithms assign the majority of subtraction operation to garbled circuits (99 of 103). The same conclusion is supported by the Euclidean distance use case. For Euclidean distance with mid-term security, both algorithms assign all 19 addition operations to garbled circuits; from the subtraction operations (with the same costs as addition), 20 of 24 are assigned to homomorphic encryption by the heuristic, and 15 by the integer program. This underpins the complexity of the context of adjacent operations and conversions.

Security	**Operator**	**LD**		**NR**		**ED**		**NL**	
	(HE|GC)	Heuristic	IP	Heuristic	IP	Heuristic	IP	Heuristic	IP
	total	375	375	302	302	75	75	268	268
short term	$\oplus$	0|99	45|54	0|0	0|0	15|4	15|4	6|93	67|32
	$\ominus$	4|99	4|99	48|12	48|12	20|4	20|4	5|1	6|0
	$\odot_e$	0|98	0|98	103|18	103|18	20|8	20|8	1|40	1|40
	$\odot_c$	0|0	0|0	0|0	0|0	0|0	0|0	2|9	2|9
	$\oslash_c$	0|0	0|0	0|1	0|1	0|0	0|0	0|9	0|9
	$\ll_c$	0|17	4|13	48|12	48|12	0|0	0|0	0|30	30|0
	$\gg_c$	0|9	0|9	0|60	0|60	0|0	0|0	0|9	0|9
	$\leq$	0|49	0|49	0|0	0|0	0|4	0|4	0|63	0|63
mid term	$\oplus$	0|99	45|54	0|0	0|0	0|19	0|19	6|93	67|32
	$\ominus$	4|99	4|99	0|60	0|60	20|4	15|9	5|1	6|0
	$\odot_e$	0|98	0|98	0|121	0|121	0|28	0|28	0|41	0|41
	$\odot_c$	0|0	0|0	0|0	0|0	0|0	0|0	2|9	2|9
	$\oslash_c$	0|0	0|0	0|1	0|1	0|0	0|0	0|9	0|9
	$\ll_c$	0|17	4|13	0|60	0|60	0|0	0|0	0|30	30|0
	$\gg_c$	0|9	0|9	0|60	0|60	0|0	0|0	0|9	0|9
	$\leq$	0|49	0|49	0|0	0|0	0|4	0|4	0|63	0|63
long term	$\oplus$	0|99	45|54	0|0	0|0	0|19	0|19	6|93	67|32
	$\ominus$	4|99	4|99	0|60	0|60	20|4	20|4	5|1	6|0
	$\odot_e$	0|98	0|98	0|121	0|121	0|28	0|28	0|41	0|41
	$\odot_c$	0|0	0|0	0|0	0|0	0|0	0|0	2|9	2|9
	$\oslash_c$	0|0	0|0	0|1	0|1	0|0	0|0	0|9	0|9
	$\ll_c$	0|17	4|13	0|60	0|60	0|0	0|0	0|30	30|0
	$\gg_c$	0|9	0|9	0|60	0|60	0|0	0|0	0|9	0|9
	$\leq$	0|49	0|49	0|0	0|0	0|4	0|4	0|63	0|63

Table 4. Operators and their protocol assignment by partitioning for algorithms long division (LD), Newton-Raphson (NR), Euclidean distance (ED), and natural logarithm (LOG) for 32 bit inputs and WAN environment. The left value shows the number of operations performed using HE and the right value those using GC, respectively.

Security	**Metric**	**LD**		**NR**		**ED**		**LOG**	
		Heuristic	IP	Heuristic	IP	Heuristic	IP	Heuristic	IP
	total	375	375	302	302	75	75	268	268
short term	GC	371	322	103	103	20	20	254	162
	HE	4	53	199	199	55	55	14	106
	$d_H(\text{GC}, \text{HE})/\text{total}$	0.011	0.141	0.659	0.659	0.733	0.733	0.052	0.396
	$d_H(\text{Heuristic}, \text{IP})/\text{total}$	0.131	0.131	0	0	0	0	0.343	0.343
	blocks	8	97	113	113	2	2	21	83
	blocks/total	0.021	0.259	0.374	0.374	0.027	0.027	0.078	0.31
	block-sizes	2	6	4	4	2	2	5	5
	largest block	6	19	26	26	55	55	219	96
mid term	GC	371	322	302	302	55	60	255	163
	HE	4	53	0	0	20	15	13	105
	$d_H(\text{GC}, \text{HE})/\text{total}$	0.011	0.141	0	0	0.267	0.2	0.049	0.392
	$d_H(\text{Heuristic}, \text{IP})/\text{total}$	0.131	0.131	0	0	0.067	0.067	0.343	0.343
	blocks	8	97	1	1	40	31	21	83
	blocks/total	0.021	0.259	0.003	0.003	0.533	0.413	0.078	0.31
	block-sizes	2	6	1	1	2	3	3	4
	largest block	6	19	1	1	2	4	219	96
long term	GC	371	322	302	302	55	55	255	163
	HE	4	53	0	0	20	20	13	105
	$d_H(\text{GC}, \text{HE})/\text{total}$	0.011	0.141	0	0	0.267	0.267	0.049	0.392
	$d_H(\text{Heuristic}, \text{IP})/\text{total}$	0.131	0.131	0	0	0	0	0.343	0.343
	blocks	8	97	1	1	40	40	21	83
	blocks/total	0.021	0.259	0.003	0.003	0.533	0.533	0.078	0.31
	block-sizes	2	6	1	1	2	2	3	4
	largest block	6	19	1	1	2	2	219	96

Table 5. Metrics and values of partitionings for algorithms long division (LD), Newton-Raphson (NR), Euclidean distance (ED), and natural logarithm (LOG) for 32 bit inputs and WAN environment. $d_H(\cdot, \cdot)$ denotes the Hamming distance between two partitionings, i.e., the number of operations that are performed with a different secure computation technique.