# A Secure Genetic Algorithm for the Subset Cover Problem and its Application to Privacy Protection

Dan Bogdanov[1], Keita Emura[2], Roman Jagomägis[1], Akira Kanaoka[3], Shin'ichiro Matsuo[2], Jan Willemson[3]

[1] Cybernetica, Mäealuse 2, 12618 Tallinn, Estonia
{dan, lighto, janwil}@cyber.ee
[2] National Institute of Information and Communications Technology, 4-2-1 Nukui-Kitamachi, Koganei, Tokyo 184-8795, Japan
{smatsuo, k-emura}@nict.go.jp
[3] Toho University, 2-2-1 Miyama, Funabashi, Chiba
akira.kanaoka@is.sci.toho-u.ac.jp

**Abstract.** We propose a method for applying genetic algorithms to confidential data. Genetic algorithms are a well-known tool for finding approximate solutions to various optimization and searching problems. More specifically, we present a secure solution for solving the subset cover problem which is formulated by a binary integer linear programming (BIP) problem (i.e. a linear programming problem, where the solution is expected to be a 0-1 vector). Our solution is based on secure multi-party computation. We give a privacy definition inspired from semantic security definitions and show how a secure computation system based on secret sharing satisfies this definition. Our solution also achieves security against timing attacks, as the execution of the secure algorithm on two different inputs is indistinguishable to the observer. We implement and benchmark our solution on the SHAREMIND secure computation system. Performance tests show that our privacy-preserving implementation achieves a 99.32% precision within 6.5 seconds on a BIP problem of moderate size. As an application of our algorithm, we consider the problem of securely outsourcing risk assessment of an end user computer environment.

**Keywords:** privacy, secure multi-party computation, genetic algorithms

## 1 Introduction

### 1.1 Background

Secure computation is a well-known cryptographic tool, where parties can jointly compute a function without revealing their own inputs. The two-party setting was introduced by Yao [34], and a more general case, secure multi-party computation (SMC), was introduced by Goldreich, Micali, and Wigderson [18].

SMC is recognized as a useful tool and several applications have been proposed, e.g. privacy-preserving data mining [26], testing disjointness of private datasets [35], applications to on-line marketplaces [13], private stable matching [33], genome-wide association studies [24], etc. Especially, due to the recent concern against cyber security incidents, it is desirable to share protection/attack knowledge, whereas such information is usually sensitive. In such a case, SMC comes into effect, e.g., privacy-preserving sharing of network monitoring data has been considered [10].

In theory, any function can be computed by garbled circuits [34] with oblivious transfer schemes which require heavy costs. However, it is a challenging task to implement an efficient SMC system, since even a simple comparison or scalar product circuit can require a few seconds to complete [29] on standard hardware.

Fully homomorphic encryption (FHE) is a new and promising technique [15]. However, the current implementations of FHE are impractical. For example, Gentry and Halevi have implemented the original Gentry's FHE scheme in [16]. In their implementation, a single bootstrapping operation (which is needed to get the complete homomorphic operation) requires at least 30 minutes (for the large setting). A FHE scheme proposed by Brakerski, Gentry, and Vaikuntanathan (which is known as FHE without bootstrapping) has also been implemented for the evaluation of the AES circuit [17]. However, one AND operation requires from 5 to 40 minutes, making the system impractical.

### 1.2 Our contribution

In this paper, we focus on solving optimization problems on confidential information. Our approach is to adapt genetic algorithms (GAs)—well-known algorithms for computing approximate solutions of the underlying problems—to SMC. GAs are inherently heuristic and are not guaranteed to produce the globally optimal result, nevertheless, they have been proven to yield results good enough for practical use. Since performance overhead added by introducing SMC is remarkable, finding good trade-offs between performance and some other parameters is an interesting research question. GA provides an interesting trade-off between precision and performance—something that has not been extensively treated in the existing literature for SMC algorithms.

We begin by defining privacy in a client-server data processing scenario. Our definition is similar to semantic security definitions for cryptosystems. We then present one secure computation setting that achieves the desired privacy goals. This setting is based on secure multiparty computation using additively homomorphic secret sharing. One of the main challenges that we tackle is security against timing attacks—our privacy definition requires that even the different executions of the secure program are indistinguishable from each other. Stating it otherwise, the program execution flow should not depend on the input data. This is a non-trivial restriction on the implementation of optimization algorithms.

We show how to securely solve (weighted) subset cover problems (SCP) formulated by binary integer linear programming (BIP) problems. We present a

BIP algorithm that satisfies our privacy definition, providing indistinguishability of any two algorithm executions. We implemented this algorithm on the Sharemind SMC system [8,6]. We provide benchmarking results from several algorithm executions with different parameters.

Finally, we consider an application of our SMC to capture the following scenario: a user who has a confidential input vector would like to solve some optimization problem in a outsourcing manner. As a concrete application, we show that our SMC can be applied for outsourced risk evaluation system [32], where it can be used to propose countermeasures that the user should deploy to reduce risks without releasing its private local information (e.g., OS/software/hardware versions).

### 1.3 Related work

Sakuma and Kobayashi [30] have proposed a secure GA for solving the distributed traveling salesman problem (TSP) in the privacy preserving manner. However, their solution uses an additively homomorphic public key encryption (e.g., the Paillier public key encryption scheme [28]) in two-party setting. Also, their approach uses Edge Assembly Crossover and is hence specific to TSP and can not directly be used to solve SCP or BIP problems.

SMC based on the Shamir secret sharing scheme was proposed by Ben-Or, Goldwasser, and Wigderson [5], and secret sharing schemes are recognized as a useful tool for constructing SMC. There are several security definitions of SMC that have been considered. In the semi-honest model, adversaries follow the protocol, but they try to extract useful information, whereas in the malicious model, adversaries can have full control over some parties who may deviate from the protocol. Recently, degradation of both security and corruptions has been considered in [22], where different security guarantees can be achieved depending on the actual number of corrupted parties. Moreover, a mixed adversary structure, where some of the parties are corrupt actively and some passively has also been recently studied [23].

As an intermediate security level between passive and active, covert adversaries can be considered [12,20,19]. In this setting the parties are willing to actively cheat, but only if they are not caught.

In addition to Sharemind [6], several SMC frameworks have been constructed so far, e.g., FairplayMP [4], VMCrypt [27], TASTY [21], SEPIA [11]. There are also other secure computation systems based on different techniques like searchable encryption, e.g. BLIND SEER [1] and CryptDB [2].

## 2 Privacy-preserving computations

### 2.1 Defining privacy for the user

Consider a distributed computation system with the following parties. $\mathcal{C}$ is the user who needs to solve a problem and uses the help of the server $\mathcal{S}$ to do that.

**Protocol 1:** Abstract protocol for server-assisted problem-solving.

```
  Data: $\mathcal{C}$ has its problem instance $e \in E$.
  Result: $\mathcal{C}$ gets a solution $m$ from the solution space $M$.
1 $\mathcal{C}$ classifies its inputs and sends them to the server:
2     $e_C \leftarrow \text{CLASSIFY}(e)$
3     $\mathcal{C}$ sends $e_C$ to $\mathcal{S}$
4 $\mathcal{S}$ solves the instance using the appropriate routine $r_C$:
5     $m_C = r_C(e_C)$
6     $\mathcal{S}$ sends $m_C$ to $\mathcal{C}$
7 $\mathcal{C}$ declassifies the solution:
8     $m \leftarrow \text{DECLASSIFY}(m_C)$
```

However, the server should not learn anything about the particular problem instance, hence we will introduce a classification mechanism that $\mathcal{C}$ can use before sending the problem out. Then $\mathcal{S}$ will solve the problem in a classified manner, and will send a similarly secured result back to $\mathcal{C}$.

For generality, we do not describe our model in the context of a particular data protection primitive such as encryption. Instead, we categorize data into *classified* and *public* categories. Classified values are confidential and must remain secret from the $\mathcal{S}$ during the computation. The *classification function* CLASSIFY() converts the public value $x$ into its classified form. The declassification function DECLASSIFY() converts a classified value to a public one. A pair of classification and declassification functions is *sound*, if

$$\forall x \; \text{DECLASSIFY}(\text{CLASSIFY}(x)) = x \,.$$

Protocol 1 presents this general setting from the perspective of the user. We assume that $\mathcal{C}$ has access to efficient classification and declassification functions.

The core of Protocol 1 is the classified computation routine $r_C$, taking the classified problem instance $e_C$ as input and generating the solution $m_C$ as output. The corresponding unclassified version of the solution routine is defined as

$$r(e) = \text{DECLASSIFY}(r_C(\text{CLASSIFY}(e)) \,.$$

We have two security requirements for the private problem solving system in Protocol 1—correctness, and oblivious execution. For *correctness*, we require that when $\mathcal{C}$ learns $m = r(e)$ at the end of the process, then $m$ is a solution to the original problem instance.

The solution routine is *oblivious*, if $\mathcal{S}$ does not learn anything about $e$ during the solving process. Note that we assume that $\mathcal{S}$ follows Protocol 1 and provides $\mathcal{C}$ with an output.

We will now give a privacy definition for the solution routine. The definition follows the real-or-random approach used in IND-CPA-style proofs (see Figure 1). We will let the attacker choose the input $e$ and later observe the transcript of the protocol consisting of the output $e_C$ of the initial CLASSIFY step and subsequent application of $r_C$. We will denote the transcript corresponding to input $e$ as $\mathsf{Transcript}(e)$. Note that by letting the adversary choose the problem instance $e$ we actually allow him to do more than we would assume in reality, where he would be a mere observer of the messages. In the accompanying random world in Figure 1, the adversary is given a transcript produced on uniformly chosen random input $e'$, and a task of distinguishing between the two worlds.

The function $\mathsf{Transcript}(e)$ is defined by the underlying implementation methodology. For example, if the system is implemented by a fully homomorphic encryption scheme, $\mathsf{Transcript}(e)$ comprises of all the values that $\mathcal{S}$ sees, whereas if the system is implemented by $k$-out-of-$n$ secret sharing (as will be done in Protocol 2), $\mathsf{Transcript}(e)$ comprises of the values seen by up to $k$ servers that make up $\mathcal{S}$. Moreover, the power of the adversary $\mathcal{A}$ is also defined according to the underlying implementation methodology. E.g., for fully homomorphic encryption, $\mathcal{A}$ should be a probabilistic polynomial-time (PPT) adversary, whereas for a secret sharing scheme, $\mathcal{A}$ is allowed to have unconditional power. As Definition 1 is an abstract privacy definition independent of the underlying technology, we need a more specific definition for each implementation. An example of an adapted privacy definition is given in Section 2.2.

```
$\mathcal{G}^{\mathcal{A}}_{\text{real}}$
    $e \leftarrow \mathcal{A}$
    return $\mathcal{A}(\mathsf{Transcript}(e))$
```

```
$\mathcal{G}^{\mathcal{A}}_{\text{rnd}}$
    $e \leftarrow \mathcal{A}$
    $e' \overset{U}{\leftarrow} E$
    return $\mathcal{A}(\mathsf{Transcript}(e'))$
```

**Fig. 1.** Privacy definition games for secure risk evaluation

**Definition 1.** *The secure solution routine in Protocol 1 is private, if for the security games in Figure 1,*

$$\Pr\left[\mathcal{G}^{\mathcal{A}}_{\text{real}} = 1\right] = \Pr\left[\mathcal{G}^{\mathcal{A}}_{\text{rnd}} = 1\right] \quad .$$

According to Definition 1, we require the constructions for CLASSIFY and $r_C$ to be such that an adversary cannot learn anything about the input $e$ of $\mathcal{C}$ by seeing $e_C$ or executing $r_C(e_C)$. We note that the adversary can not have access to the declassification oracle, as it would then be trivial to break privacy. The particular secure computation technique used for implementing the solution routine will have to provide constructions for CLASSIFY, DECLASSIFY and $r_C$ so that this assumption holds.

## 2.2 A threshold version of the privacy definition

In Section 2.1 we gave a definition for privacy, but omitted details on how to achieve it. In this section, we will describe how to satisfy this definition using

**Protocol 2:** Server-assisted problem-solving using SMC.

```
Data: $\mathcal{C}$ has its problem instance $e \in E$.
Result: $\mathcal{C}$ gets a solution $m$ from the solution space $M$.
1 $\mathcal{C}$ shares its problem instance and sends shares to the solving servers:
2     $(e_1, \ldots, e_n) \leftarrow \mathsf{Share}(e)$
3     $\mathcal{C}$ sends $e_i$ to $\mathcal{S}_i$
4 Each $\mathcal{S}_i$ participates in SMC to find the solution:
5     $(m_1, \ldots, m_n) = r(e_1, \ldots, e_n)$
6     $\mathcal{S}_i$ sends $m_i$ to $\mathcal{C}$
7 $\mathcal{C}$ reconstructs the solutions from shares:
8     $m \leftarrow \mathsf{Reconstruct}(m_1, \ldots, m_n)$
```

secret sharing and secure multi-party computation. Alternatively, one could build a secure problem solving system using, for example, homomorphic encryption, garbled circuits or trusted hardware.

SMC requires that we implement the routine as a distributed system, but the same holds for most other cryptographic techniques. Fully homomorphic encryption could be used to create a single-server solution in theory, but currently known protocols are very inefficient in practice [16]. Trusted hardware that provides data protection and anti-tamper guarantees would also be a suitable tool for implementing $r_C$. However, such hardware is still not widely available.

We give an updated privacy definition that allows the problem solving system to be distributed between $n$ parties $\mathcal{S}_1, \ldots, \mathcal{S}_n$. First, we define the CLASSIFY and DECLASSIFY functions using secret sharing [31]. To classify a value $s$, we compute its shares $s_1, \ldots, s_n$ using the sharing function of the chosen secret sharing scheme and send $s_i$ to $\mathcal{S}_i$. Similarly, to declassify a value, each $\mathcal{S}_i$ must send its share of the value to $\mathcal{C}$. The updated protocol is given as Protocol 2.

We use the threshold notion in our security assumption. Namely, we assume that no more than $k$ nodes in the problem-solving system $\mathcal{S}$ are corrupted by the adversary. In the context of Definition 1 based on the games in Figure 1 this means that the $\mathsf{Transcript}$ available to the adversary will consist of the views of up to $k$ nodes.

To prove that the adversary cannot distinguish between the real and random game, we need to show that in the $\mathsf{Transcript}$, the adversary cannot distinguish 1) the shares of its chosen input from the shares of random input, and 2) the computations performed on these shared inputs. In the next section we will present one specific instantiation of all the required components allowing for the required security proofs.

## 2.3 The Sharemind secure multi-party computation platform

In order to implement the components CLASSIFY, DECLASSIFY and $r_C$, we need to instantiate the abstract secret sharing and share computing engine with a concrete one. For the purposes of this paper, we chose the Sharemind framework for implementing privacy-preserving computations [6]. Sharemind supports the operations that we require for our risk analysis task and the protocols are universally composable, simplifying our privacy proof. SHAREMIND was chosen for its performance and rapid application development tools [7,9]. Also, we could obtain the software implementation of SHAREMIND for conducting practical experiments.

In its current implementation, SHAREMIND uses three computing nodes (also called *miners*) working on additively shared 32-bit unsigned integers. To share a value $x \in \mathbb{Z}_{2^{32}}$, two random elements $x_1, x_2 \in \mathbb{Z}_{2^{32}}$ are generated and the third (uniquely determined) value $x_3 \in \mathbb{Z}_{2^{32}}$ is selected so that $x_1 + x_2 + x_3 \equiv x \bmod 2^{32}$. This is essentially the definition for the CLASSIFY operation. The corresponding DECLASSIFY operation is even simpler – the three shares just need to be added modulo $2^{32}$.

In order to implement the private problem solving function $r_C$ we need a number of primitive protocols for addition, multiplication, compare-exchange, etc. The specifications of these protocols and the security proofs can be found in [9,25].

All the SHAREMIND protocols have been designed to withstand a passive attacker who is able to monitor the communications of one computing node out of three. For such an attacker, the Transcript of the protocol will not differ from a random Transcript. More formally, we are using the definition of perfect simulatability given in [6].

**Definition 2.** *We say that an SMC protocol is perfectly simulatable if there exists an efficient universal non-rewinding simulator $S$ that can simulate all protocol messages to any real-world adversary $\mathcal{A}$ so that for all input shares, the output distributions of $\mathcal{A}$ and $S(\mathcal{A})$ coincide.*

It is additionally proved in [6] that if a perfectly simulatable protocol is appended by a perfectly secure resharing step, we obtain a perfectly secure protocol. Using this result, the paper [9] proves that all the fundamental protocols (multiplication, share conversion, bit extraction, equality, division) used by the current SHAREMIND engine are secure against one passive adversary. Furthermore, due to universal composability of perfectly secure elementary operations, all the fundamental protocols also remain universally composable. This implies that higher level protocols (such as sorting) retain the property of having the Transcript indistinguishable from the random one. Hence the requirements of Definition 1 are satisfied as long as no intermediate results are declassified. Achieving this presumes that the protocols implemented are data-agnostic, that is, the program flow of the algorithm does not depend on its inputs. Unfortunately, not all efficient algorithms are data-agnostic and we need to select or design algorithms based on this quality. In the following section we will present one approach to implementing a data-agnostic private optimization problem solving function.

## 3 Privacy-preserving optimization problem solving

In this Section, we will concentrate on the cheapest subset covering problem. Formally, let $\mathcal{Z} = \{z_1, z_2, \ldots, z_m\}$ be a set of $m$ elements, and let $X_1, X_2, \ldots, X_n \subseteq$

$\mathcal{Z}$ be a collection of available subsets of $\mathcal{Z}$. Let us also have a target set $T \subseteq \mathcal{Z}$ and our aim is to select some $X_{i_j}$ out of the given collection so that the selected sets would cover $T$, i.e.

$$T \subseteq \bigcup_j X_{i_j} \; . \tag{1}$$

Additionally, let every set $X_i$ have an associated cost $c_i$; then our optimization goal is

$$\sum_j c_j \to \min \; . \tag{2}$$

We will represent the covering restrictions in terms of the incidence matrix $A = (a_{ij})_{i,j=1}^{m,n}$ given by

$$a_{ij} = \begin{cases} 1 & \text{if } z_i \in X_j, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

We represent the target set $T$ by its characteristic vector $\mathbf{t} = (t_1, t_2, \ldots, t_m)^T$ where $t_i = 1$ indicates that $z_i \in T$. As the output, we are required to produce another 0-1 vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, where $x_j = 1$ indicates that the set $X_j$ was selected.

Then the condition (1) translates to

$$A \times \mathbf{x} \geq \mathbf{t} \tag{3}$$

and the optimization goal (2) becomes

$$\mathbf{c} \cdot \mathbf{x} = \sum_{j=1}^{n} c_j \cdot x_j \to \min \, ,$$

where $\mathbf{c} = (c_1, c_2, \ldots, c_n)^T$.

Out of the data given to the algorithm, the matrix $A$ and the cost vector $\mathbf{c}$ are assumed to be public, but the vectors $\mathbf{t}$ and $\mathbf{x}$ must remain oblivious. Formulated as such, we have a standard binary integer programming problem (BIP) that have been well-studied and can be solved by branch-and-bound type of algorithms like Balas Additive Algorithm [3].

However, in order to efficiently prune the search tree, such methods need to make decisions on control bits, and their runs differ on different input data. This behaviour is unwanted in a privacy-preserving algorithm, as the running time of the program could be used to infer details about the private inputs.

Hence, we decided not to choose a branch-and-bound algorithm and take a totally different approach. In this paper, we implement a genetic algorithm for solving the underlying BIP problem. This approach has several advantages. First, we do not have to leak any bits, since the control flow does not depend on the private inputs. Second, a genetic algorithm can be made to run for a predefined number of iterations or a predefined amount of time. On the other hand, genetic algorithms are inherently heuristic and are not guaranteed to produce

**Algorithm 3:** Basic genetic algorithm

```
Data: Characteristic vector $\mathbf{t} \in \{0,1\}^m$; incidence matrix $A \in \{0,1\}^{m \times n}$; vector
      $\mathbf{c} \in (\mathbb{Z}_{2^{32}})^n$ expressing the costs of subsets
Result: A set of $k$ candidate solutions
1  Generate a random generation $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$
2  while there is time to compute do
3      For each pair of individuals $\mathbf{x}_i$ and $\mathbf{x}_j$ produce their offspring by
       crossover
4      For some offspring mutate some of their bits
5      Sort the offspring pool by the fitness $\mathbf{c} \cdot \mathbf{x}$
6      Choose $k$ fittest for the new generation $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$.
7  end
8  return $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$
```

the globally optimal result. Nevertheless, they have been proven to yield results good enough for practical use.

Genetic algorithms work on *generations* of individuals. In our case, the individuals are 0-1 vectors $\mathbf{x}_i$ corresponding to the candidate countermeasure suites. Each generation has $k$ individuals where $k$ is a system-wide configurable parameter. Computation proceeds in iterations, where both the input and output of each iteration is a $k$-element generation. The general structure of the routine is presented in Algorithm 3.

There are several implementation details to fill in in the basic algorithm. We have to choose the size of the generation, crossover strategy and mutation strategy. Since these parameters depend on each other non-linearly, making the optimal choices is a highly non-trivial task.

For our demo application we ran tests with the size of the generation set to $k = 8, 12, 16, 23, 32$, and for the number of iterations $g = 5, 10, 20$. We applied uniform crossover and mutated the bits of individuals also randomly with the probability $2^{-s}$. The last two choices were made because of the need to hide the control flow. Next to the uniform crossover, another frequently used strategy is one- or two-point crossover. However, selecting a few random cutting points has no straightforward implementation in the oblivious setting. At the same time, uniform selection between the parent genes is rather easy to achieve by generating random selection vectors and performing $n$ oblivious choices for each candidate offspring. Similar reasoning applies to the mutation operation as well. In order to flip the bits of individuals with probability $2^{-s}$, we can generate $s$ random bit vectors and multiply them bitwise. In our experiments we set $s = 4$ giving 6.25% of probability for any bit to be flipped.

The pool of candidates for the next generation consists of $k$ members of the previous generation plus $\binom{k}{2}$ of their offspring. Since technically, it is simpler to sort $2^t$ elements, some of the offspring are discarded to get the closest power of two for the pool size. E.g. when $k = 8$, we get the original pool size $8 + \binom{8}{2} = 36$

and we drop 4 of them to get down to 32. For $k = 12, 16, 23, 32$, we sort arrays of size $64, 128, 256, 512$, respectively.

In order to select the $k$ fittest individuals, i.e. the candidate covers with the smallest cost, several steps need to be taken. First, for every candidate vector we need to verify the matrix inequality in Equation 3, and if it is not satisfied, we obliviously assign a very high cost to this vector. Next, we need to sort all the candidate vectors by the costs. Full sorting is a rather expensive operation, and it is not really needed for the purposes of genetic algorithms. Hence, we decided to implement Swiss tournament sorting. It is known that this sorting method works better in both ends on the sorted array, whereas the middle part is not guaranteed to be linearly ordered [14]. In our case, we obliviously evaluate as much of the Swiss tournament sorting network as is needed for finding the top $k$ elements. However, our experiments show that compared to full sorting, the degradation of the precision of the whole genetic algorithm is rather small, but the gain in computing time is significant.

To conclude the Section, we state and prove the main theorem of the paper.

**Theorem 1.** *Algorithm 3 is private in the sense of Definition 1.*

*Proof.* The proof relies on two main building blocks – privacy of the primitive operations and preservation of the privacy property through composition.

In order to implement Algorithm 3, only two primitive operations are needed – addition and multiplication.

Indeed, generating a random initial bitvector for the first generation is a trivial local operation. Crossover can be implemented by multiplying $s$ random bitvectors and then applying oblivious choice as specified in [8]. Mutation operation also needs a biased random vector which can be generated as in the case of crossover, and then applying an XOR operation that can be implemented as

$$a \operatorname{XOR} b = a + b - 2ab\,.$$

Fitness computation is a simple dot product which only needs addition and multiplication. For sorting, a greater-than primitive and a compare-exchange block are needed. Suitable constructions are found in [8] and [25]. Note that Swiss tournament sorting can be implemented as a sorting network and is hence data agnostic, i.e. the control flow does not depend on the actual values.

Addition of values is a local operation and trivially satisfies Definition 1. A suitable multiplication together with the accompanying privacy proof is given in [9]. The necessary compositionality theorems are given in [6]. This completes the proof.

Next, we will present a concrete application scenario for our optimization framework.

## 4 Application scenario: secure service provisioning framework

Our example scenario builds on top of the problem of outsourcing risk assessment computations. In [32] Takahashi *et al.* have proposed the concept of a risk visualisation and alerting framework. The framework consists of four components.

The **user system** contains the platform and applications utilized by the user requesting access to an online service. We assume that the user system connects to services over a network and can collect information about its software, hardware and network connection. The **service provider** is providing users with a service over a network. Each service provider can set security requirements for using the service.

The **security authority** collects information about threats to software, hardware and networking systems and the respective countermeasures to compile a knowledge base. This knowledge base is used by the **risk evaluation system** (RES) to help the user system in selecting appropriate countermeasures for securing online transactions.

When a user decides to access an online service, the user system compiles a description of its environment and sends it to the risk evaluation system together with the security expectations. The risk evaluation system determines the security threats that could affect the user's transaction and proposes countermeasures that the user should deploy to reduce risks.

The service provision framework is illustrated in Figure 2. We refer to [32] for more details.

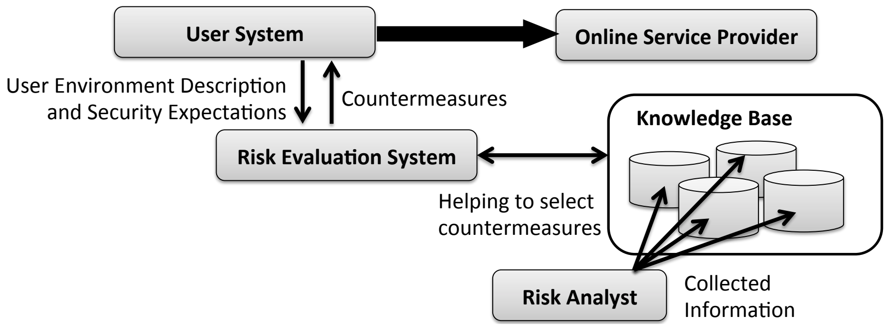

**Fig. 2.** The secure service provision framework.

While the use of this framework enhances the security of online transactions that the user performs, a naïve implementation does so at the expense of privacy, since the user is forced to disclose the information about its vulnerability status to a third, potentially untrusted party.

However, this problem can be solved by applying our secure subset cover computation routine. In terms defined in Section 3, we may view $\mathcal{Z} = \{z_1, z_2, \ldots, z_m\}$ as the set of possible threats against the user system. The sets $X_i$ correspond to the possible countermeasures, where $z_j \in X_i$ (i.e. $a_{ij} = 1$) means that the measure $X_i$ is efficient against the threat $z_j$. The input characteristic vector $\mathbf{t}$ then refers to all the threats relevant for the particular user, and the output vector $\mathbf{x}$ describes a set of countermeasures that, in collection, mitigate all the threats and are together as cheap as the system was able to find within given time. In order to complete real performance tests, we built a model problem, the parameters of which can be found in Appendix A.

Note that our test vectors do not reflect any real environment, and are provided for benchmarking purposes only. The actual parameter values may vary between different real setups and need to be re-evaluated as a part of real risk analysis process. This work remains out of the scope of the current paper.

## 5 Practical results

We implemented the risk evaluation system on the Sharemind system. We created a data importer that was used to load the knowledge base into a Sharemind installation. We then developed the described genetic algorithm in the SecreC programming language that is used to implement Sharemind applications. We implemented the oblivious top-$k$ as a new protocol in Sharemind for optimization purposes. We then created a testing application that let Sharemind evaluate the risks on all possible inputs according to the used knowledge base. We also computed all the optimal solutions using Sage and the GNU Linear Programming Kit and used the results as reference values to evaluate the correctness of the private implementation.

The Sharemind system was deployed on three computers running on a local network. The computers contain 12 CPU cores and 48 gigabytes of RAM. However, during experiments we saw that at most two cores per machine were being fully used and the memory usage of Sharemind did not grow over 150 megabytes. It is reasonable to assume such resources, as the Risk Evaluation System will be deployed centrally, on high-performance hardware.

For our performance tests, we selected $m = 10$ threats and $n = 16$ countermeasures together with their correspondences and costs as described in Appendix A. We ran the tests for generation sizes $k = 8, 12, 16, 23, 32$ and number of generations $g = 5, 10, 20$. For each of the pairs of these values, we determined the percentage of correctly computed optimal costs out of $2^{10} = 1024$ possible input vectors. We also measured the average execution time. The results are displayed in Table 1.

We see that in already under 6.5 seconds it is possible to achieve near-perfect performance of the algorithm, and that increasing the size of the generation helps to obtain better precision with much lower cost in time compared to increasing the number of generations.

|  | $g = 5$ | $g = 10$ | $g = 20$ |
|---|---|---|---|
| $k = 8$ | 3.71% (3711 ms) | 45.21% (7187 ms) | 78.22% (14167 ms) |
| $k = 12$ | 18.75% (4220 ms) | 79.39% (8186 ms) | 92.87% (16120 ms) |
| $k = 16$ | 55.66% (4733 ms) | 95.61% (9247 ms) | 99.51% (18291 ms) |
| $k = 23$ | 89.55% (5420 ms) | 99.80% (10546 ms) | 99.90% (21008 ms) |
| $k = 32$ | 99.32% (6440 ms) | 100.00% (12702 ms) | 100.00% (25164 ms) |

**Table 1.** Accuracy and running time of the privacy-preserving genetic algorithm for $g$ generations of size $k$.

## 6 Conclusions and future works

After the first introduction of the SMC concept in early 1980s, continuous research efforts have been carried out to take this concept to practical applications. The current paper also contributes to this research.

One of the main problems when trying to implement practical SMC systems is the prohibitive performance overhead. This paper considers one possible trade-off to address this problem, namely relaxing the precision requirements in order to achieve better running time of the algorithms. One setting where such a trade-off makes sense are the optimization problems. Even then, not all the optimization methods are suitable for implementing using SMC mechanisms. Most notably, the method should be data-agnostic.

In this paper, we considered weighted subset covering problems and constructed a genetic algorithm to solve them. We implemented this algorithm on top of the Sharemind SMC engine and benchmarked on the model problem of secure outsourced risk analysis. Our results show that on moderate size problems, genetic algorithm running on Sharemind can have excellent precision in reasonably fast running time, with many possible trade-offs.

Genetic algorithms are by far not the only method for solving optimization problems. It is an interesting future research target to develop privacy-preserving versions of other well-known approaches (gradient descent, simulated annealing, ant colony optimization, etc.).

On the other hand, weighted subset covering problem is rather general and it has many other possible application areas (e.g. determining suitable treatment for a patient without revealing his/her exact medical condition). Deploying our algorithms to solve these problems and improving their performance remain the subjects for future research as well.

## Acknowledgements

This research has been supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science (EXCS), UaESMC project financed by the EU 7th Framework Programme under grant agreement no. 284731, and Estonian Research Council through grant IUT27-1.

## References

1. BLIND SEER: Bloom index search of encrypted results. `http://www.cs.columbia.edu/nsl/projects/blind_seer/`.
2. CryptDB. `http://css.csail.mit.edu/cryptdb/`.
3. E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13(4):517–546, July/August 1965.
4. A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM CCS '08*, pages 257–266, 2008.
5. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC '88*, pages 1–10, 1988.
6. D. Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.
7. D. Bogdanov, R. Jagomägis, and S. Laur. A Universal Toolkit for Cryptographically Secure Privacy-Preserving Data Mining. In *PAISI '12*,, volume 7299 of *LNCS*, pages 112–126. Springer, 2012.
8. D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *ESORICS '08*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.
9. D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6):403–418, 2012.
10. J.-M. Bohli, W. Li, and J. Seedorf. Assisting server for secure multi-party computation. In *WISTP '12*, pages 144–159, 2012.
11. M. Burkhart, M. Strasser, D. Many, and X. A. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX '10*, pages 223–240, 2010.
12. N. Chandran, V. Goyal, R. Ostrovsky, and A. Sahai. Covert multi-party computation. In *FOCS '07*, pages 238–248, 2007.
13. S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *CT-RSA*, pages 416–432, 2012.
14. W. Elmenreich, T. Ibounig, and I. Fehérvári. Robustness versus performance in sorting and tournament algorithms. *Acta Polytechnica Hungarica*, 6(5):7–18, 2009.
15. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09*, pages 169–178, 2009.
16. C. Gentry and S. Halevi. Implementing Gentry's Fully-Homomorphic Encryption Scheme. In *EUROCRYPT '11*, pages 129–148, 2011.
17. C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO '12*, pages 850–867, 2012.
18. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC '87*, pages 218–229, 1987.
19. V. Goyal and A. Jain. On the round complexity of covert computation. In *STOC '10*, pages 191–200, 2010.
20. V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *EUROCRYPT '06*, pages 289–306, 2008.
21. W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In *ACM CCS '10*, pages 451–462, 2010.

22. M. Hirt, C. Lucas, U. Maurer, and D. Raub. Graceful degradation in multi-party computation (extended abstract). In *ICITS '11*, pages 163–180, 2011.
23. M. Hirt, C. Lucas, U. Maurer, and D. Raub. Passive corruption in statistical multi-party computation - (extended abstract). In *ICITS '12*, pages 129–146, 2012.
24. L. Kamm, D. Bogdanov, S. Laur, and J. Vilo. A new way to protect privacy in large-scale genome-wide association studies. *Bioinformatics*, 29(7):886–893, 2013.
25. S. Laur, J. Willemson, and B. Zhang. Round-Efficient Oblivious Database Manipulation. In *ISC '11*, volume 7001 of *LNCS*, pages 262–277, 2011.
26. Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.
27. L. Malka. VMCrypt: modular software architecture for scalable secure computation. In *ACM CCS '11*, pages 715–724, 2011.
28. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99*, pages 223–238, 1999.
29. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *ASIACRYPT '09*, pages 250–267, 2009.
30. J. Sakuma and S. Kobayashi. A genetic algorithm for privacy preserving combinatorial optimization. In *GECCO '07*, pages 1372–1379, 2007.
31. A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.
32. T. Takahashi, K. Emura, A. Kanaoka, S. Matsuo, and T. Minowa. Risk visualization and alerting system: Architecture and proof-of-concept implementation. In *SESP '13*, pages 3–10. ACM, 2013.
33. T. Teruya and J. Sakuma. Round-efficient private stable matching from additive homomorphic encryption. In *ISC '13*, 2014, to appear.
34. A. C.-C. Yao. Protocols for secure computations (extended abstract). In *FOCS '82*, pages 160–164, 1982.
35. Q. Ye, H. Wang, J. Pieprzyk, and X.-M. Zhang. Efficient disjointness tests for private datasets. In *ACISP '08*, pages 155–169, 2008.

## A Description of the experimental setup

For our tests, we selected $m = 10$ threats and $n = 16$ countermeasures together with their correspondences and costs, having some typical network services in mind (e.g. social networking service, on-line banking, electronic commerce, and on-line storage service). We considered the following threats:

T1. Authentication Information Leakage from Terminal Inside
T2. Authentication Information Leakage by Shoulder Surfing
T3. Authentication Information Leakage on Data Transmission Channel (LAN)
T4. Authentication Information Leakage on Data Transmission Channel (End-to-End)
T5. Platform Information Leakage from User Terminal
T6. Privacy Information Leakage from User Terminal
T7. Privacy Information Leakage on Data Transmission Channel
T8. Classified Information Leakage from User Terminal
T9. Disable Services
T10. Financial Damage.

Against these threats we considered the following countermeasures:

C1. Authentication: Password (stored in Client Terminal)
C2. Authentication: Password (short length, not stored)
C3. Authentication: Password (long length, not stored)
C4. Authentication: Challenge and Response
C5. Authentication: Look-Up Table
C6. Authentication: Software Cryptographic Token
C7. Authentication: Hardware Cryptographic Token
C8. Anti-Virus Gateway
C9. Anti-Virus Client
C10. Channel Encryption (LAN)
C11. Channel Encryption (End-to-End)
C12. Stored Data Encryption
C13. Digital Signature
C14. Firewall
C15. Intrusion Detection System (IDS) / Intrusion Prevention System (IPS)
C16. Proxy

Based on the expert knowledge of the authors, we then selected the matrix of correspondence between the threats and countermeasures (see Table 2) and the vector of countermeasure costs (see Table 3).

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| T2 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T3 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| T4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| T5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| T6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| T7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| T8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| T9 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T10 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |

**Table 2.** Test data for matrix of threats and countermeasures

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cost | 3 | 1 | 2 | 5 | 7 | 15 | 30 | 150 | 15 | 3 | 5 | 15 | 15 | 5 | 100 | 100 |

**Table 3.** Test data for countermeasure costs