# Secure Evaluation of Private Linear Branching Programs with Medical Applications

Mauro Barni[1], Pierluigi Failla[1], Vladimir Kolesnikov[2], Riccardo Lazzeretti[1], Ahmad-Reza Sadeghi[3], and Thomas Schneider[3]

[1] Department of Information Engineering, University of Siena, Italy
barni@dii.unisi.it, {pierluigi.failla,lazzaro79}@gmail.com*
[2] Bell Laboratories, 600 Mountain Ave. Murray Hill, NJ 07974, USA
kolesnikov@research.bell-labs.com
[3] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
{ahmad.sadeghi,thomas.schneider}@trust.rub.de**

**Abstract.** Diagnostic and classification algorithms play an important role in data analysis, with applications in areas such as health care, fault diagnostics, or benchmarking. Branching programs (BP) is a popular representation model for describing the underlying classification/diagnostics algorithms. Typical application scenarios involve a client who provides data and a service provider (server) whose diagnostic program is run on client's data. Both parties need to keep their inputs private.

We present new, more efficient privacy-protecting protocols for remote evaluation of such classification/diagnostic programs. In addition to efficiency improvements, we generalize previous solutions – we securely evaluate private linear branching programs (LBP), a useful generalization of BP that we introduce. We show practicality of our solutions: we apply our protocols to the privacy-preserving classification of medical ElectroCardioGram (ECG) signals and present implementation results. Finally, we discover and fix a subtle security weakness of the most recent remote diagnostic proposal, which allowed malicious clients to learn partial information about the program.

## 1 Introduction

Classification and diagnostic programs are very useful tools for automatic data analysis with respect to specific properties. They are deployed for various applications, from spam filters [8], remote software fault diagnostics [12] to medical diagnostic expert systems [29]. The health-care industry is moving faster than ever toward technologies that offer personalized online self-service, medical error reduction, consumer data mining and more (e.g., [11]). Such technologies have the potential of revolutionizing the way medical data is stored, processed, delivered, and made available in an ubiquitous and seamless way to millions of users all over the world.

* Supported by EU FP6 project SPEED and MIUR project 2007JXH7ET.
** Supported by EU FP6 project SPEED and EU FP7 project CACE.

Typical application scenarios in this context concern two (remote) parties, a user or data provider (client) and a service provider (server) who usually owns the diagnostic software that will run on the client's data and output classification/diagnostic results.

In this framework, however, a central problem is the protection of privacy of both parties. On the one hand, the user's data might be sensitive and security-critical (e.g., electronic patient records in health care, passwords and other secret credentials in remote software diagnostics, trade- and work-flow information in benchmarking of enterprises). On the other hand, the service provider, who owns the diagnostic software, may not be willing to disclose the underlying algorithms and the corresponding optimized parameters (e.g., because they represent intellectual property).

Secure function evaluation with private functions [31,27,18,30] is one way to realize the above scenarios, when the underlying private algorithms are represented as circuits. However, as we elaborate in the discussion on related work, in some applications, such as diagnostics, it is most natural and efficient to represent the function as a decision graph or a Branching Program (BP). At a high level, BPs consist of different types of nodes — decision nodes and classification nodes. Based on the inputs and certain decision parameters such as thresholds (that are often the result of learning processes), the algorithm branches among the decision nodes until it reaches the corresponding classification node (which represents a leaf node in the decision tree).

In this work, we consider applications that benefit from the BP representation, such as our motivating application, classification of medical ElectroCardioGram (ECG) signals. In the remainder of the paper, we concentrate on the BP approach (including discussion of related work).

**Related Work.** There is a number of fundamental works, e.g. Kilian [16], that rely on Branching Programs (BP) "under the hood". These are general feasibility results that do not attempt to achieve high efficiency for concrete problems. The goals and results of these works and ours are different. We do not directly compare their performance to ours; instead, we compare our work with previously-best approaches that are applicable to our setting (see below).

Recently, very interesting BP-based crypto-computing protocols were proposed by Ishai and Paskin [14] (and later slightly improved by Lipmaa [22] who also presented a variety of applications). In their setting, the server evaluates his program on client's encrypted data. The novelty of the approach of [14] is that the communication and client's computation depend on the length (or depth) of BP, and are *independent* of the size of BP. This allows for significant savings in cases of "wide" BP. However, the protocol requires computationally expensive operations on homomorphically encrypted ciphertexts for each node of the BP. Further, the server's computation still depends on the size of BP. The savings achieved by these protocols are not significant in our setting (in applications we are considering, BPs are not wide), and the cost of employed homomorphic encryption operation outweighs the benefit.

Most relevant for this work is the sequence of works [19,4,32], where the authors consider problems similar to ours, and are specifically concerned with concrete performance of the resulting protocols. Kruger et al. [19] observed that some functions are more succinctly represented by Ordered Binary Decision Diagrams (OBDD), and proposed a natural extension of the garbled circuit method which allows secure evaluation of (publicly known) OBDDs. As in the garbled circuit approach, the client receives garblings of his inputs, and is blindly evaluating a garbled OBDD to receive a garbling of the output, which is then opened. Brickell et al. [4] further extended this approach and considered evaluation of private BPs. They also consider a more complex decision procedure at the nodes of BP (based on the result of integer comparison). The solution of [4] is especially suited for remote diagnostics, their motivating application.

In the above two approaches the communication complexity depends linearly on the size of the BP, as the size of the garbled BP is linear in the size of the BP. While the computational complexity for the client remains asymptotically the same as in the crypto-computing protocols of [14] (linear in the length of the evaluation path), the computational cost is substantially smaller (especially for the server), as only symmetric crypto operations need to be applied to the nodes of the BP. In [32] an extension of the protocol of [19] for secure evaluation of private OBDDs based on efficient selection blocks [18] was proposed. In our work, we generalize, unify, extend, and improve efficiency of the above three protocols [19,4,32].

In addition to circuits and BPs, other (secure) classification methods have been considered, such as those based on neural networks [6,25,28,30]. In our work, we concentrate on the BP representation.

**Our Contribution and Outline.** Our main contribution is a new more efficient modular protocol for secure evaluation of a class of diagnostics/classification problems, which are naturally computed by (a generalization of) decision trees (§3). We work in the semi-honest model, but explain how our protocols can be efficiently secured against malicious adversaries (§3.6). We improve on the previously proposed solutions in several ways. Firstly, we consider a more general problem. It turns out, our motivating example — ECG classification — as well as a variety of other applications, benefit from a natural generalization of Branching Programs (BP) and decision trees, commonly considered before. We introduce and justify *Linear Branching Programs* (LBP) (§3.1), and show how to evaluate them efficiently. Secondly, we fine-tune the performance. We propose several new tricks (for example, we show how to avoid inclusion of classification nodes in the encrypted program). We also employ performance-improving techniques which were used in a variety of areas of secure computation. This results in significant performance improvements over previous work, even for evaluation of previously considered BPs. A detailed performance comparison is presented in §3.5. Further, in §4, we discover and fix a subtle vulnerability in the recent and very efficient variant of the protocol for secure BP evaluation [4] and secure classifier learning [5]. Finally, we apply our protocols to the privacy-preserving classification of medical ElectroCardioGram (ECG) signals (§5).

## 2 Preliminaries

In our protocols we combine several standard cryptographic tools (additively homomorphic encryption, oblivious transfer, and garbled circuits) which we summarize in §2.1. Readers familiar with these tools can safely skip §2.1 and continue reading our notational conventions in §2.2.

We denote the symmetric (asymmetric) security parameter with $t$ ($T$). Recommended sizes for short-term security are $t = 80, T = 1248$ [10].

### 2.1 Cryptographic Tools

**Homomorphic Encryption (HE).** We use a semantically secure additively homomorphic public-key encryption scheme. In an additively homomorphic cryptosystem, given encryptions $[\![a]\!]$ and $[\![b]\!]$, an encryption $[\![a+b]\!]$ can be computed as $[\![a+b]\!] = [\![a]\!][\![b]\!]$, where all operations are performed in the corresponding plaintext or ciphertext structure. From this property follows, that multiplication of an encryption $[\![a]\!]$ with a constant $c$ can be computed efficiently as $[\![c \cdot a]\!] = [\![a]\!]^c$ (e.g., with the square-and-multiply method). As instantiation we use the Paillier cryptosystem [26,7] which has plaintext space $\mathbb{Z}_N$ and ciphertext space $\mathbb{Z}^*_{N^2}$, where $N$ is a $T$-bit RSA modulus. This scheme is semantically secure under the decisional composite residuosity assumption (DCRA). For details on the encryption and decryption function we refer to [7].

**Parallel Oblivious Transfer (OT).** Parallel 1-out-of-2 Oblivious Transfer for $m$ bitstrings of bitlength $\ell$, denoted as $\mathsf{OT}^m_\ell$, is a two-party protocol. $\mathcal{S}$ inputs $m$ pairs of $\ell$-bit strings $S_i = \langle s^0_i, s^1_i \rangle$ for $i = 1, .., m$ with $s^0_i, s^1_i \in \{0,1\}^\ell$. $\mathcal{C}$ inputs $m$ choice bits $b_i \in \{0,1\}$. At the end of the protocol, $\mathcal{C}$ learns $s^{b_i}_i$, but nothing about $s^{1-b_i}_i$ whereas $\mathcal{S}$ learns nothing about $b_i$. We use $\mathsf{OT}^m_\ell$ as a black-box primitive in our constructions. It can be instantiated efficiently with different protocols [24,2,21,13]. Extensions of [13] can be used to reduce the number of computationally expensive public-key operations to be independent of $m$. We omit the parameters $m$ or $\ell$ if they are clear from the context.

**Garbled Circuit (GC).** Yao's Garbled Circuit approach [33], excellently presented in [20], is the most efficient method for secure evaluation of a boolean circuit $C$. We summarize its ideas in the following. First, the circuit **constructor** (server $\mathcal{S}$), creates a *garbled circuit* $\widetilde{C}$ with algorithm $\mathsf{CreateGC}$: for each wire $W_i$ of the circuit, he randomly chooses a *complementary garbled value* $\widetilde{W_i} = \langle \widetilde{w}^0_i, \widetilde{w}^1_i \rangle$ consisting of two secrets, $\widetilde{w}^0_i$ and $\widetilde{w}^1_i$, where $\widetilde{w}^j_i$ is the *garbled value* of $W_i$'s value $j$. (Note: $\widetilde{w}^j_i$ does not reveal $j$.) Further, for each gate $G_i$, $\mathcal{S}$ creates and sends to the **evaluator** (client $\mathcal{C}$) a *garbled table* $\widetilde{T}_i$ with the following property: given a set of garbled values of $G_i$'s inputs, $\widetilde{T}_i$ allows to recover the garbled value of the corresponding $G_i$'s output, and nothing else. Then garbled values corresponding to $\mathcal{C}$'s inputs $x_j$ are (obliviously) transferred to $\mathcal{C}$ with a parallel oblivious transfer protocol $\mathsf{OT}$: $\mathcal{S}$ inputs complementary garbled values $\widetilde{W_j}$ into the protocol; $\mathcal{C}$ inputs $x_j$ and obtains $\widetilde{w}^{x_j}_j$ as outputs. Now, $\mathcal{C}$ can evaluate the garbled circuit $\widetilde{C}$ with algorithm $\mathsf{EvalGC}$ to obtain the garbled output simply by evaluating the garbled circuit gate by gate, using the garbled tables $\widetilde{T}_i$. Correctness of GC follows from method of construction of garbled tables $\widetilde{T}_i$. As in [4] we use the GC protocol as a conditional oblivious transfer protocol where we do not provide a translation from the garbled output values to their plain values to $\mathcal{C}$, i.e., $\mathcal{C}$ obtains one of two garbled values which can be used as key in subsequent protocols but does not know to which value this key corresponds.

*Implementation Details.* A *point-and-permute technique* can be used to speed up the implementation of the GC protocol [23]: The garbled values $\widetilde{w}_i = \langle k_i, \pi_i \rangle$ consist of a symmetric key $k_i \in \{0,1\}^t$ and $\pi_i \in \{0,1\}$ is a random permutation bit. The permutation bit $\pi_i$ is used to select the right table entry for decryption with the key $k_i$. Extensions of [17] to "free XOR" gates can be used to further improve performance of GC.

### 2.2 Notation

**Number Representation.** In the following, a *(signed) $\ell$-bit integer* $x^\ell$ is represented as one bit for the sign, $sign(x^\ell)$, and $\ell - 1$ bits for the magnitude, $abs(x^\ell)$, i.e., $-2^{\ell-1} < x^\ell < +2^{\ell-1}$. This allows *sign-magnitude representation* of numbers in a circuit, i.e., one bit for the sign and $\ell - 1$ bits for the magnitude. For homomorphic encryptions we use *ring representation*, i.e., $x^\ell$ with $2^\ell \le N$ is mapped into an element of the plaintext group $\mathbb{Z}_N$ using $m(x^\ell) = \begin{cases} x^\ell, & \text{if } x^\ell \ge 0 \\ N + x^\ell, & \text{if } x^\ell < 0 \end{cases}$.

**Homomophic Encryption.** $\mathsf{Gen}(1^T)$ denotes the key generation algorithm of the Paillier cryptosystem [26,7] which, on input the asymmetric security parameter $T$, outputs secret key $sk_\mathcal{C}$ and public key $pk_\mathcal{C} = N$ to $\mathcal{C}$, where $N$ is a $T$-bit RSA modulus. $[\![x^\ell]\!]$ denotes the encryption of an $\ell$-bit message $x^\ell \in \mathbb{Z}_N$ (we assume $\ell < T$) with public key $pk_\mathcal{C}$.

**Garbled Objects.** Objects overlined with a tilde symbol denote garbled objects: Intuitively, $\mathcal{C}$ cannot infer the real value $i$ from a garbled value $\widetilde{w}^i$, but can use garbled values to evaluate a garbled circuit $\widetilde{C}$ or a garbled LBP $\widetilde{\mathcal{L}}$. Capital letters $\widetilde{W}$ denote complementary garbled values consisting of two garbled values $\left\langle \widetilde{w}^0, \widetilde{w}^1 \right\rangle$ for which we use the corresponding small letters. We group together multiple garbled values to a *garbled $\ell$-bit value* $\widetilde{\mathbf{w}}^\ell$ (small, bold letter) which consists of $\ell$ garbled values $\widetilde{w}_1, \ldots, \widetilde{w}_\ell$. Analogously, a *complementary garbled $\ell$-bit value* $\widetilde{\mathbf{W}}^\ell$ (capital, bold letter) consists of $\ell$ complementary garbled values $\widetilde{W}_1, \ldots, \widetilde{W}_\ell$.

## 3 Evaluation of Private Linear Branching Programs

After formally defining Linear Branching Programs (LBP) in §3.1, we present two protocols for secure evaluation of private LBPs. We decompose our protocols into different building blocks similar to the protocol of [4] and show how to instantiate them more efficiently than in [4].

The protocols for secure evaluation of private LBPs are executed between a server $\mathcal{S}$ in possession of a private LBP, and a client $\mathcal{C}$ in possession of data, called **attribute vector**. Let $z$ be the number of nodes in the LBP, and $n$ be the number of attributes in the attribute vector.

As in most practical scenarios $n$ is significantly larger than $z$, the protocol of [4] is optimized for this case. In particular, the size of our securely transformed LBP depends linearly on $z$ but is independent of $n$.

In contrast to [4], our solutions do not reveal the total number $z$ of nodes of the LBP, but only its number of decision nodes $d$ for efficiency improvements. In particular, the size of our securely transformed LBP depends linearly on $d$ which is smaller than $z$ by up to a factor of two.

## 3.1 Linear Branching Programs (LBP)

First, we formally define the notion of linear branching programs. We do so by generalizing the BP definition used in [4]. We note that BPs – and hence also LBPs – generalize binary classification or decision trees and Ordered Binary Decision Diagrams (OBDDs) used in [19,32].

**Definition 1 (Linear Branching Program).** *Let* $\mathbf{x}^\ell = x_1^\ell, .., x_n^\ell$ *be the* **attribute vector** *of signed $\ell$-bit integer values. A binary* **Linear Branching Program (LBP)** $\mathcal{L}$ *is a triple* $\langle \{P_1, .., P_z\}, \mathit{Left}, \mathit{Right} \rangle$. *The first element is a set of $z$ nodes consisting of $d$* **decision nodes** $P_1, .., P_d$ *followed by $z - d$* **classification nodes** $P_{d+1}, .., P_z$.
*Decision nodes $P_i$, $1 \le i \le d$ are the internal nodes of the LBP. Each* $P_i := \left\langle \mathbf{a_i^\ell}, t_i^{\ell'} \right\rangle$ *is a pair, where* $\mathbf{a_i^\ell} = \left\langle a_{i,1}^\ell, .., a_{i,n}^\ell \right\rangle$ *is the* **linear combination vector** *consisting of $n$ signed $\ell$-bit integer values and $t_i^{\ell'}$ is the signed $\ell'$-bit integer* **threshold** *value with which* $\mathbf{a_i^\ell} \circ \mathbf{x}^\ell = \sum_{j=1}^n a_{i,j}^\ell x_j^\ell$ *is compared in this node.* *$\mathit{Left}(i)$ is the index of the next node if* $\mathbf{a_i^\ell} \circ \mathbf{x}^\ell \le t_i^{\ell'}$; *$\mathit{Right}(i)$ is the index of the next node if* $\mathbf{a_i^\ell} \circ \mathbf{x}^\ell > t_i^{\ell'}$. *Functions $\mathit{Left}()$ and $\mathit{Right}()$ are such that the resulting directed graph is acyclic.*
*Classification nodes $P_j := \langle c_j \rangle$, $d < j \le z$ are the leaf nodes of the LBP consisting of a single classification label $c_j$ each.*

To evaluate the LBP $\mathcal{L}$ on attribute vector $\mathbf{x}^\ell$, start with the first decision node $P_1$. If $\mathbf{a_1^\ell} \circ \mathbf{x}^\ell \le t_1^{\ell'}$, move to node *Left*(1), else to *Right*(1). Repeat this process recursively (with corresponding $\mathbf{a_i^\ell}$ and $t_i^{\ell'}$), until reaching one of the classification nodes and obtaining the classification $c = \mathcal{L}(\mathbf{x}^\ell)$.

In the general case of LBPs, the bit-length $\ell'$ has to be chosen according to the maximum value of linear combinations as $\ell' = 2\ell + \lceil \log_2 n \rceil - 1$.
As noted above, LBPs can be seen as a generalization of previous representations:

- **Branching Programs (BP)** as used in [4] are a special case of LBPs. In a BP, in each decision node $P_i$ the $\alpha_i$-th input $x_{\alpha_i}^\ell$ is compared with the threshold value $t_i^{\ell'}$, where $\alpha_i \in \{0,..,n\}$ is a private index. In this case, the linear combination vector $\mathbf{a_i^\ell}$ of the LBP decision node degrades to a **selection vector** $\mathbf{a_i} = \langle a_{i,1},..,a_{i,n}\rangle$, with exactly one entry $a_{i,\alpha_i} = 1$ and all other entries $a_{i,j\neq\alpha_i} = 0$. The bit-length of the threshold values $t_i^{\ell'}$ is set to $\ell' = \ell$.

– **Ordered Binary Decision Diagrams (OBDD)** as used in [19,32] are a special case of BPs with bit inputs ($\ell = 1$) and exactly two classification nodes ($P_{z-1} = \langle 0\rangle$ and $P_z = \langle 1\rangle$).

## 3.2 Protocol Overview

We start with a high-level overview of our protocol for secure evaluation of private linear branching programs. We then fill in the technical details and outline the differences and improvements of our protocol over previous work in the following sections.

Our protocol SecureEvalPrivateLBP, its main building blocks, and the data and communication flows are shown in Fig. 1. The client $\mathcal{C}$ receives an attribute vector $\mathbf{x}^\ell = \{x_1^\ell, \ldots, x_n^\ell\}$ as input, and the server $\mathcal{S}$ receives a linear branching program $\mathcal{L}$. Upon completion of the protocol, $\mathcal{C}$ outputs the classification label $c = \mathcal{L}(\mathbf{x}^\ell)$, and $\mathcal{S}$ learns nothing. Of course, both $\mathcal{C}$ and $\mathcal{S}$ wish to keep their inputs private. Protocol SecureEvalPrivateLBP is naturally decomposed into the following three phases (cf. Fig. 1).

CreateGarbledLBP. In this phase, $\mathcal{S}$ creates a garbled version of the LBP $\mathcal{L}$. This is done similarly to the garbled-circuit-based previous approaches [4,19]. The idea is to randomly permute the LBP, encrypt the pointers on the left and right successor, and garble the nodes, so that the evaluator is unable to deviate from the evaluation path defined by his input.

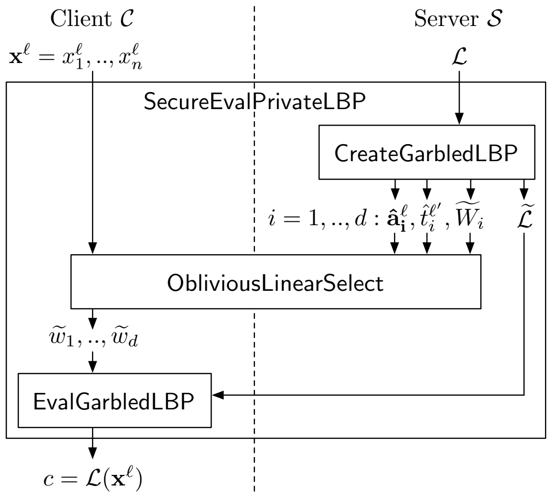

Client $\mathcal{C}$
Server $\mathcal{S}$
$\mathbf{x}^\ell = x_1^\ell,..,x_n^\ell$
$\mathcal{L}$
SecureEvalPrivateLBP
CreateGarbledLBP
$i = 1,..,d : \hat{\mathbf{a}}_i^\ell, \hat{t}_i^{\ell'}, \widetilde{W}_i$
$\widetilde{\mathcal{L}}$
ObliviousLinearSelect
$\widetilde{w}_1,..,\widetilde{w}_d$
EvalGarbledLBP
$c = \mathcal{L}(\mathbf{x}^\ell)$

**Fig. 1.** Secure Evaluation of Private Linear Branching Programs - Structural Overview

The novelty of our solution is that each node transition is based on the oblivious comparison of a *linear combination of inputs with a node-specific threshold.* Thus, CreateGarbledLBP additionally processes (and modifies) these values and passes them to the next phase. CreateGarbledLBP can be entirely precomputed by $\mathcal{S}$.

ObliviousLinearSelect. In this phase, $\mathcal{C}$ obliviously obtains the garbled values $\widetilde{w}_1, .., \widetilde{w}_d$ which correspond to the outcome of the comparisons of the linear combination of the attribute vector with the threshold for each garbled node. These garbled values will then be used to evaluate the garbled LBP in the next phase. Making analogy to Yao's garbled circuit (GC), this phase is the equivalent of the GC evaluator receiving the wire secrets corresponding to his inputs. In our protocol, this stage is more complicated, since the secrets are transferred based on secret conditions.

EvalGarbledLBP. This phase is equivalent to Yao's GC evaluation. Here, $\mathcal{C}$ receives the garbled LBP $\widetilde{\mathcal{L}}$ from $\mathcal{S}$, and evaluates it. EvalGarbledLBP additionally gets the garbled values $\widetilde{w}_1, .., \widetilde{w}_d$ output by ObliviousLinearSelect as inputs and outputs the classification label $c = \mathcal{L}(\mathbf{x}^\ell)$.

## 3.3 Our Building Blocks

**Phase I (offline): CreateGarbledLBP.** In this pre-computation phase, $\mathcal{S}$ generates a garbled version $\widetilde{\mathcal{L}}$ of the private branching program $\mathcal{L}$. CreateGarbledLBP is presented in Algorithm 1.

Algorithm CreateGarbledLBP converts the nodes $P_i$ of $\mathcal{L}$ into garbled nodes $\widetilde{P}_i$ in $\widetilde{\mathcal{L}}$, as follows. First, we associate a randomly chosen key $\Delta_i$ with each node $P_i$. We use $\Delta_i$ (with other keys, see below) for encryption of $P_i$'s data. Each decision node $P_i$ contains a pointer to its left successor node $P_{i_0}$ and one to its right successor node $P_{i_1}$. Garbled $\widetilde{P}_i$ contains encryptions of these pointers and of successors' respective keys $\Delta_{i_0}, \Delta_{i_1}$. Further, since we want to prevent the LBP evaluator from following both successor nodes, we additionally separately encrypt the data needed to decrypt $P_{i_0}$ and $P_{i_1}$ with random keys $k_i^0$ and $k_i^1$ respectively. Evaluator later will receive (one of) $k_i^j$, depending on his input (see block ObliviousLinearSelect), which will enable him to decrypt and follow only the corresponding successor node. The used *semantically secure symmetric encryption* scheme can be instantiated as $\mathsf{Enc}_k^s(m) = m \oplus H(k||s) = \mathsf{Dec}_k^s(m)$, where $s$ is a unique identifier used once, and $H(k||s)$ is a pseudorandom function (PRF) evaluated on $s$ and keyed with $k$, e.g., a cryptographic hash function from the SHA-2 family. In CreateGarbledLBP, we use the following technical improvement from [19]: Instead of encrypting twice (sequentially, with $\Delta_i$ and $k_i^j$), we encrypt successor $P_{i_j}$'s data with $\Delta_i \oplus k_i^j$. Each classification node is garbled simply by including its label directly into the parent's node (instead of the decryption key $\Delta_i$). This eliminates the need for inclusion of classification nodes in the garbled LBP and increases the size of each garbled decision node by only two bits denoting the type of its successor nodes. This

**Algorithm 1.** CreateGarbledLBP

```
Input S: LBP $\mathcal{L} = \langle\{P_1,..,P_z\}, Left, Right\rangle$. For $i \le d$, $P_i$ is a decision node $\left\langle \mathbf{a_i^\ell}, t_i^{\ell'} \right\rangle$.
    For $i > d$, $P_i$ is a classification node $\langle c_i \rangle$.
Output S: (i) Garbled LBP $\widetilde{\mathcal{L}} = \left\langle \{\widetilde{P}_1,..,\widetilde{P}_d\} \right\rangle$; (ii) Compl. garbled inputs $\widetilde{W}_1,..,\widetilde{W}_d$;
    (iii) Perm. lin. comb. vectors $\hat{\mathbf{a}}_\mathbf{1}^\ell,..,\hat{\mathbf{a}}_\mathbf{d}^\ell$; (iv) Perm. thresholds $\hat{t}_1^{\ell'},..,\hat{t}_d^{\ell'}$
 1: choose a random permutation $\Pi$ of the set $1,..,d$ with $\Pi[1] = 1$.
 2: choose key $\Delta_1 := 0^t$, rand. keys $\Delta_i \in_R \{0,1\}^t$, $1 < i \le d$ for enc. decision nodes
 3: for $i = 1$ to $d$ do $\{P_i = \left\langle \mathbf{a_i^\ell}, t_i^{\ell'} \right\rangle$ is a decision node$\}$
 4:    let permuted index $\hat{i} := \Pi[i]$
 5:    set perm. linear combination vector $\hat{\mathbf{a}}_{\hat{\mathbf{i}}}^\ell := \mathbf{a_i^\ell}$; perm. threshold value $\hat{t}_{\hat{i}}^{\ell'} := t_i^{\ell'}$
 6:    choose rand. compl. garbled value $\widetilde{W}_{\hat{i}} = \langle \widetilde{w}_{\hat{i}}^0 = \langle k_{\hat{i}}^0, \pi_{\hat{i}} \rangle, \widetilde{w}_{\hat{i}}^1 = \langle k_{\hat{i}}^1, 1 - \pi_{\hat{i}} \rangle \rangle$
 7:    let left successor $i_0 := Left[i]$, $\hat{i}_0 := \Pi[i_0]$ (permuted)
 8:    if $i_0 \le d$ then $\{P_{i_0}$ is a decision node$\}$
 9:       let $m^{\hat{i},0} := \left\langle \texttt{"decision"}, \hat{i}_0, \Delta_{\hat{i}_0} \right\rangle$
10:    else $\{P_{i_0} = \langle c_{i_0} \rangle$ is a classification node$\}$
11:       let $m^{\hat{i},0} := \langle \texttt{"classification"}, c_{i_0} \rangle$
12:    end if
13:    let right successor $i_1 := Right[i]$, $\hat{i}_1 := \Pi[i_1]$ (permuted)
14:    if $i_1 \le d$ then $\{P_{i_1}$ is a decision node$\}$
15:       let $m^{\hat{i},1} := \left\langle \texttt{"decision"}, \hat{i}_1, \Delta_{\hat{i}_1} \right\rangle$
16:    else $\{P_{i_1} = \langle c_{i_1} \rangle$ is a classification node$\}$
17:       let $m^{\hat{i},1} := \langle \texttt{"classification"}, c_{i_1} \rangle$
18:    end if
19:    let garbled decision node $\widetilde{P}_{\hat{i}} := \left\langle Enc^{\hat{i},0}_{k_{\hat{i}}^{\pi_{\hat{i}}} \oplus \Delta_{\hat{i}}}(m^{\hat{i},\pi_{\hat{i}}}), Enc^{\hat{i},1}_{k_{\hat{i}}^{1-\pi_{\hat{i}}} \oplus \Delta_{\hat{i}}}(m^{\hat{i},1-\pi_{\hat{i}}}) \right\rangle$
20: end for
21: return $\widetilde{\mathcal{L}} := \left\langle \{\widetilde{P}_1,..,\widetilde{P}_d\} \right\rangle; \widetilde{W}_1,..,\widetilde{W}_d; \hat{\mathbf{a}}_\mathbf{1}^\ell,..,\hat{\mathbf{a}}_\mathbf{d}^\ell; \hat{t}_1^{\ell'},..,\hat{t}_d^{\ell'}$
```

technical improvement allows to reduce the size of the garbled LBP by up to a factor of 2, depending on the number of classification nodes. Finally, the two successors' encryptions are randomly permuted.

We note that sometimes the order of nodes in a LBP may leak some information. To avoid this, in the garbling process we randomly permute the nodes of the LBP (which results in the corresponding substitutions in the encrypted pointers). The start node $P_1$ remains the first node in $\widetilde{\mathcal{L}}$. Additionally, garbled nodes are padded s.t. they all have the same size.

The output of CreateGarbledLBP is $\widetilde{\mathcal{L}}$ (to be sent to $\mathcal{C}$), and the randomness used in its construction (to be used by $\mathcal{S}$ in the next phase).

*Complexity (cf. Table 2).* $\widetilde{\mathcal{L}}$ contains $d$ garbled nodes $\widetilde{P}_i$ consisting of two ciphertexts of size $\lceil \log d \rceil + t + 1$ bits each (assuming classification labels $c_j$ have less bits than this). The asymptotic size of $\widetilde{\mathcal{L}}$ is $2d(\log d + t)$ bits.

*Tiny LBPs.* In case of tiny LBPs with a small number of decision nodes $d$ we describe an alternative construction method for garbled LBPs with asymptotic size $2^d \log(z-d)$ in the full version of this paper [3].

**Phase II: ObliviousLinearSelect.** In this phase, $\mathcal{C}$ obliviously obtains the garbled values $\widetilde{w}_1, .., \widetilde{w}_d$ which correspond to the outcome of the comparison of the linear combination of the attribute vector with the threshold for each garbled node. These garbled values will then be used to evaluate the garbled LBP $\widetilde{\mathcal{L}}$ in the next phase.

In ObliviousLinearSelect, the input of $\mathcal{C}$ is the private attribute vector $\mathbf{x}^\ell$ and $\mathcal{S}$ inputs the private outputs of CreateGarbledLBP: complementary garbled values $\widetilde{W}_1 = \langle \widetilde{w}_1^0, \widetilde{w}_1^1 \rangle, .., \widetilde{W}_d = \langle \widetilde{w}_d^0, \widetilde{w}_d^1 \rangle$, permuted linear combination vectors $\hat{\mathbf{a}}_\mathbf{1}^\ell, .., \hat{\mathbf{a}}_\mathbf{d}^\ell$, and permuted threshold values $\hat{t}_1^{\ell'}, .., \hat{t}_d^{\ell'}$. Upon completion of the ObliviousLinearSelect protocol, $\mathcal{C}$ obtains the garbled values $\widetilde{w}_1, .., \widetilde{w}_d$, as follows: if $\hat{\mathbf{a}}_\mathbf{i}^\ell \circ \mathbf{x}^\ell > \hat{t}_i^{\ell'}$, then $\widetilde{w}_i = \widetilde{w}_i^1$; else $\widetilde{w}_i = \widetilde{w}_i^0$. $\mathcal{S}$ learns nothing about $\mathcal{C}$'s inputs.

We give two efficient instantiations for ObliviousLinearSelect in §3.4.

**Phase III: EvalGarbledLBP.** In the last phase, $\mathcal{C}$ receives the garbled LBP $\widetilde{\mathcal{L}}$ from $\mathcal{S}$, and evaluates it locally with algorithm EvalGarbledLBP as shown in Algorithm 2. This algorithm additionally gets the garbled values $\widetilde{w}_1, .., \widetilde{w}_d$ output by ObliviousLinearSelect as inputs and outputs the classification label $c = \mathcal{L}(\mathbf{x}^\ell)$.

**Algorithm 2.** EvalGarbledLBP

```
Input C: (i) Garbled LBP $\widetilde{\mathcal{L}} = \left\langle \{\widetilde{P}_1, .., \widetilde{P}_d\} \right\rangle$; (ii) Garbled input values $\widetilde{w}_1, .., \widetilde{w}_d$
Output C: Classification label $c$ such that $c = \mathcal{L}(\mathbf{x}^\ell)$

1: let $\hat{i} := 1; \Delta_{\hat{i}} := 0^t$ (start at root)
2: while true do
3:     let $\langle k_{\hat{i}}, \pi_{\hat{i}} \rangle := \widetilde{w}_{\hat{i}}$; $\langle c_{\hat{i}}^0, c_{\hat{i}}^1 \rangle := \widetilde{P}_{\hat{i}}$; $\langle \text{type}_{\hat{i}}, \text{data}_{\hat{i}} \rangle := \text{Dec}_{k_{\hat{i}} \oplus \Delta_{\hat{i}}}^{\hat{i}, \pi_{\hat{i}}}(c_{\hat{i}}^\pi)$
4:     if $\text{type}_{\hat{i}} = \texttt{"decision"}$ then
5:         let $\left\langle \hat{i}, \Delta_{\hat{i}} \right\rangle := \text{data}_{\hat{i}}$
6:     else
7:         let $\langle c \rangle := \text{data}_{\hat{i}}$
8:         return $c$
9:     end if
10: end while
```

$\mathcal{C}$ traverses the garbled LBP $\widetilde{\mathcal{L}}$ by decrypting garbled decision nodes along the evaluation path starting at $\widetilde{P}_1$. At each node $\widetilde{P}_{\hat{i}}$,[1] $\mathcal{C}$ takes the garbled attribute value $\widetilde{w}_{\hat{i}} = \langle k_{\hat{i}}, \pi_{\hat{i}} \rangle$ together with the node-specific key $\Delta_{\hat{i}}$ to decrypt the information needed to continue evaluation of the garbled successor node until the correct classification label $c$ is obtained.

[1] We use the permuted index $\hat{i}$ here to stress that $\mathcal{C}$ does not obtain any information from the order of garbled nodes.

It is easy to see that some information about $\mathcal{L}$ is leaked to $\mathcal{C}$, namely: (i) the total number $d$ of *decision* nodes in the program $\widetilde{\mathcal{L}}$, and (ii) the length of the evaluation path, i.e., the number of decision nodes that have been evaluated before reaching the classification node. We note that in many cases this is acceptable. If not, this information can be hidden using appropriate padding of $\mathcal{L}$. We further note that $\widetilde{\mathcal{L}}$ cannot be reused. Each secure evaluation requires construction of a new garbled LBP.

## 3.4 Oblivious Linear Selection Protocol

We show how to instantiate the ObliviousLinearSelect protocol next.

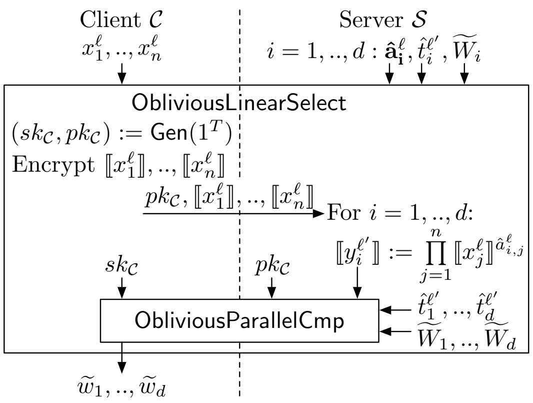
A straight-forward instantiation can be obtained by evaluating a garbled *circuit* whose size depends on the number of attributes $n$. This construction is described in the full version of this paper [3].

In the following, we concentrate on an alternative instantiation based on a *hybrid* combination of homomorphic encryption and garbled circuits which results in a better communication complexity.

**Hybrid Instantiation.** In this instantiation of ObliviousLinearSelect (see Fig. 2 for an overview), $\mathcal{C}$ generates a key-pair for the additively homomorphic encryption scheme and sends the public key $pk_{\mathcal{C}}$ together with the homomorphically encrypted attributes $[\![x_1^\ell]\!], .., [\![x_n^\ell]\!]$ to $\mathcal{S}$. Using the additively homomorphic property, $\mathcal{S}$ can compute the linear combination of these ciphertexts with the private coefficients $\hat{\mathbf{a}}_{\mathbf{i}}^\ell$ as $[\![y_i^{\ell'}]\!] := [\![\sum_{j=1}^n \hat{a}_{i,j}^\ell x_j^\ell]\!] = \prod_{j=1}^n [\![x_j^\ell]\!]^{\hat{a}_{i,j}^\ell}$, $1 \le i \le d$. Afterwards, the encrypted values $[\![y_i^{\ell'}]\!]$ are obliviously compared with the threshold values $\hat{t}_i^{\ell'}$ in the ObliviousParallelCmp protocol. This protocol allows $\mathcal{C}$ to obliviously obtain the garbled values corresponding to the comparison of $y_i^{\ell'}$ and $\hat{t}_i^{\ell'}$, i.e., $\widetilde{w}_i^0$ if $y_i^{\ell'} \le \hat{t}_i^{\ell'}$ and $\widetilde{w}_i^1$ otherwise. ObliviousParallelCmp ensures that neither $\mathcal{C}$ nor $\mathcal{S}$ learns anything about the plaintexts $y_i^{\ell'}$ from which they could deduce information about the other party's private function or inputs.

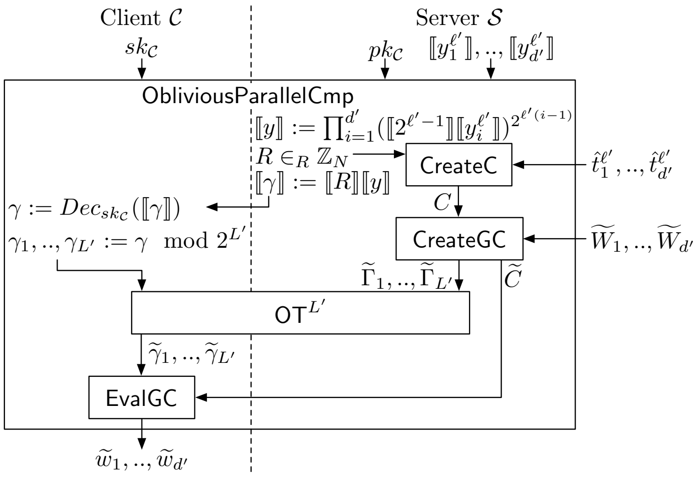
ObliviousParallelCmp *protocol (cf. Fig. 3).* The basic idea underlying this protocol is that $\mathcal{S}$ blinds the encrypted value $[\![y_i^{\ell'}]\!]$ in order to hide the encrypted plaintext from $\mathcal{C}$. To achieve this, $\mathcal{S}$ adds a randomly chosen value $R \in_R \mathbb{Z}_N$[2] under encryption before sending them to $\mathcal{C}$ who can decrypt but does not learn the plain value. Afterwards, a garbled circuit $C$ is evaluated which obliviously takes off the blinding value $R$ and compares the result (which corresponds to $y_i^{\ell'}$) with the threshold value $t_i^{\ell'}$. We improve the communication complexity of this basic protocol which essentially corresponds to the protocol of [4] by packing together multiple ciphertexts and minimizing the size of the garbled circuit as detailed in the full version of this paper [3]. The complexity of our improved protocol is given in Table 1.

[2] In contrast to [4], we choose $R$ from the full plaintext space in order to protect against malicious behavior of $\mathcal{C}$ as explained in §4.



Client $\mathcal{C}$
$x_1^\ell, .., x_n^\ell$
Server $\mathcal{S}$
$i = 1, .., d : \hat{\mathbf{a}}_i^\ell, \hat{t}_i^{\ell'}, \widetilde{W}_i$
ObliviousLinearSelect
$(sk_\mathcal{C}, pk_\mathcal{C}) := \mathsf{Gen}(1^T)$
Encrypt $[\![x_1^\ell]\!], .., [\![x_n^\ell]\!]$
$pk_\mathcal{C}, [\![x_1^\ell]\!], .., [\![x_n^\ell]\!]$
For $i = 1, .., d$:
$[\![y_i^{\ell'}]\!] := \prod_{j=1}^{n} [\![x_j^\ell]\!]^{\hat{a}_{i,j}^\ell}$
$sk_\mathcal{C}$
$pk_\mathcal{C}$
ObliviousParallelCmp
$\hat{t}_1^{\ell'}, .., \hat{t}_d^{\ell'}$
$\widetilde{W}_1, .., \widetilde{W}_d$
$\widetilde{w}_1, .., \widetilde{w}_d$

**Fig. 2.** ObliviousLinearSelect - Hybrid



Client $\mathcal{C}$
$sk_\mathcal{C}$
Server $\mathcal{S}$
$pk_\mathcal{C}$
$[\![y_1^{\ell'}]\!], .., [\![y_{d'}^{\ell'}]\!]$
ObliviousParallelCmp
$[\![y]\!] := \prod_{i=1}^{d'} ([\![2^{\ell'-1}]\!][\![y_i^{\ell'}]\!])^{2^{\ell'(i-1)}}$
$R \in_R \mathbb{Z}_N$
CreateC
$\hat{t}_1^{\ell'}, .., \hat{t}_{d'}^{\ell'}$
$[\![\gamma]\!] := [\![R]\!][\![y]\!]$
$\gamma := Dec_{sk_\mathcal{C}}([\![\gamma]\!])$
$C$
$\gamma_1, .., \gamma_{L'} := \gamma \mod 2^{L'}$
CreateGC
$\widetilde{W}_1, .., \widetilde{W}_{d'}$
$\widetilde{\Gamma}_1, .., \widetilde{\Gamma}_{L'}$
$\widetilde{C}$
$\mathsf{OT}^{L'}$
$\widetilde{\gamma}_1, .., \widetilde{\gamma}_{L'}$
EvalGC
$\widetilde{w}_1, .., \widetilde{w}_{d'}$

**Fig. 3.** ObliviousParallelCmp

We note that further performance improvements can be achieved when the client only computes those values he will actually use in the LBP evaluation phase ("lazy evaluation"). All server-visible messages of OT must be performed to hide the evaluation path taken based on client's inputs.

**Extension of [4] to LBPs.** Our hybrid instantiation of the ObliviousLinearSelect protocol is a generalization of the ObliviousAttributeSelection protocol proposed in [4]. The protocol for secure evaluation of private BPs of [4] can easily be extended to a protocol for secure evaluation of private LBPs by computing a linear combination of the ciphertexts instead of obliviously selecting one ciphertext. We call this protocol "ext. [4]". However, our hybrid protocol is more efficient than ext. [4] as shown in the following.

## 3.5 Performance Improvements over Existing Solutions

On the one hand, our protocols for secure evaluation of private LBPs extend the functionality that can be evaluated securely from OBDDs [19], private OBDDs [32], and private BPs [4] to the larger class of private LBPs. On the other hand, our protocols can be seen as general protocols which simply become improved (more efficient) versions of the protocols of [19,32,4] when instantiated for the respective special case functionality.

The employed techniques and the resulting performance improvements of our protocols over previous solutions (see Table 1 and Table 2) are summarized in the full version of this paper [3].

**Table 1.** Protocols for Secure Evaluation of Private BPs/LBPs with parameters $z$: #nodes, $d$: #decision nodes, $n$: #attributes, $\ell$: bitlength of attributes, $\ell'$: bitlength of thresholds (for LBPs), $t$: symmetric security parameter, $T$: asymmetric security parameter, $\kappa$: statistical correctness parameter

| Oblivious Selection Protocol | Private Function | Moves | Asymptotic Communication Complexity GC | OT | HE |
|---|---|---|---|---|---|
| [4] | BP | $\mathrm{OT}+2$ | $12z\ell(t+\kappa)$ | $\mathrm{OT}_t^{z\ell}$ | $(n+z)2T$ |
| ext. [4] (§3.4) | LBP | | $12z\ell'(t+\kappa)$ | $\mathrm{OT}_t^{z\ell'}$ | |
| our Hybrid (§3.4) | BP | $\mathrm{OT}+2$ | $12d\ell t$ | $\mathrm{OT}_t^{d\ell}$ | $(n+\frac{\ell}{T-\kappa}d)2T$ |
| | LBP | | $12d\ell' t$ | $\mathrm{OT}_t^{d\ell'}$ | $(n+\frac{\ell'}{T-\kappa}d)2T$ |
| our Circuit [3] | BP | OT | $4(n\log d+3d\log d)\ell t$ | $\mathrm{OT}_t^{n\ell}$ | |
| | LBP | | $16nd(\ell^2+\ell')t$ | | |

**Table 2.** Algorithms to Create/Evaluate Garbled LBPs. Parameters as in Table 1.

| Algorithm to Create/Evaluate Garbled LBP | Size of Garbled LBP in bit | Examples from [4] with $t=80,\kappa=80$ | | |
|---|---|---|---|---|
| | | iptables $d=4, z=9$ | mpg321 $d=5, z=9$ | nfs $d=12, z=17$ |
| [19,4] | $2z(\lceil\log z\rceil+t+\kappa)$ | 2,952 bit | 2,952 bit | 5,610 bit |
| Alg. 1 & 2 (§3.3) | $2d(\lceil\log d\rceil+t+1)$ | 664 bit | 840 bit | 2,040 bit |
| Tiny GLBP [3] | $2^d\lceil\log(z-d)\rceil$ | 48 bit | 64 bit | 12,288 bit |

## 3.6 Correctness and Security Properties

As previously mentioned, protocol SecureEvalPrivateLBP securely and correctly evaluates private LBP in the semi-honest model. We formally state and prove the corresponding theorems in the full version of this paper [3].

**Extensions to Malicious Players.** We note that our protocols, although proven secure against semi-honest players, tolerate many malicious client behaviors. For example, many efficient OT protocols are secure against malicious chooser, and a malicious client is unable to tamper with the GC evaluation procedure. Further, our protocols can be modified to achieve full security in the malicious model. One classical way is to prove in zero-knowledge the validity of every step a party takes. However, this approach is far inefficient. We achieve malicious security simply by employing efficient sub-protocols proven secure against malicious players. (This is the transformation approach suggested in [4].) More specifically, we use committed OT, secure two-party computation on committed inputs, and verifiable homomorphic encryption schemes (see [15] for more detailed description).

## 4 A Technical Omission in [4] w.r.t. Malicious Client

In this section, we briefly present and fix a small technical omission, which led to an incorrect claim of security in the setting with semi-honest server and malicious client in [4, Section 4.4] (and indirectly propagated to [5]). Recall, the protocol of [4] is similar in the structure to our protocol. The problem appears in the ObliviousAttributeSelection subroutine, which is similar to (actually is a special case of) our ObliviousLinearSelect subroutine. The issue is that, for efficiency, [4] mask the $\mathcal{C}$-encrypted attribute values with relatively short random strings, before returning them back to $\mathcal{C}$. In the semi-honest model this guarantees that $\mathcal{C}$ is not able to match the returned strings to the attribute values he earlier sent, and the security of the entire protocol holds. However, the security breaks in case of a malicious $\mathcal{C}$. Indeed, such a $\mathcal{C}$ can send $\mathcal{S}$ very large values $x_i$, wait for the blinded responses and match these with the original $x_i$, allowing $\mathcal{C}$ to determine which of the attributes are used for the computation. (Indeed, whereas the lower bits are blinded correctly, the upper bits of the maliciously chosen large $x_i$ remain the same.) We further note that malicious $\mathcal{C}$ will not even be caught since he will recover the blinding values and will be able to continue execution with his real inputs, if he wishes.

This attack can be prevented by choosing $R$ randomly from the full plaintext domain $\mathbb{Z}_N$ instead (as done in our ObliviousParallelCmp protocol). With this modification, the blinded value is entirely random in $\mathbb{Z}_N$ and a malicious $\mathcal{C}$ cannot infer any information from it.

## 5 Application: Secure Classification of Medical Data

Our motivating example application for secure evaluation of private LBPs is privacy-preserving classification of biomedical data. As a simple representative example we consider privacy-preserving classification of ElectroCardioGram (ECG) signals. A patient (client $\mathcal{C}$) owns an ECG signal and asks a service provider (server $\mathcal{S}$) to determine which class the ECG signal belongs to. $\mathcal{C}$ requires $\mathcal{S}$ to gain no knowledge about the ECG signal (as this is sensitive personal data of $\mathcal{C}$), whereas $\mathcal{S}$ requires no disclosure of details of the classification algorithm to $\mathcal{C}$ (as this represents valuable intellectual property of $\mathcal{S}$). We show how tho achieve this by mapping an established ECG classification algorithm [1,9] to secure evaluation of a private LBP, and give implementation results in the full version of this paper [3].

**Acknowledgments.** We thank anonymous reviewers of ESORICS 2009 for their helpful comments.

## References

1. Acharya, U.R., Suri, J., Spaan, J.A.E., Krishnan, S.M.: Advances in Cardiac Signal Processing, ch. 8. Springer, Heidelberg (2007)
2. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (2001)
3. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.-R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications (Full Version). Cryptology ePrint Archive, Report 2009/195 (2009)
4. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: ACM CCS 2007, pp. 498–507. ACM Press, New York (2007)
5. Brickell, J., Shmatikov, V.: Privacy-preserving classifier learning. In: FC 2009. LNCS. Springer, Heidelberg (2009)
6. Chang, Y.-C., Lu, C.-J.: Oblivious polynomial evaluation and oblivious neural learning. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 369–384. Springer, Heidelberg (2001)
7. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Kim, K.-c. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001)
8. Delany, S.J., Cunningham, P., Doyle, D., Zamolotskikh, A.: Generating estimates of classification confidence for a case-based spam filter. In: Muñoz-Ávila, H., Ricci, F. (eds.) ICCBR 2005. LNCS (LNAI), vol. 3620, pp. 177–190. Springer, Heidelberg (2005)
9. Ge, D.F., Srinivasan, N., Krishnan, S.M.: Cardiac arrhythmia classification using autoregressive modeling. BioMedical Engineering OnLine 1(1), 5 (2002)
10. Giry, D., Quisquater, J.-J.: Cryptographic key length recommendation (March 2009), `http://keylength.com`
11. Google Health (2009), `https://www.google.com/health`
12. Ha, J., Rossbach, C.J., Davis, J.V., Roy, I., Ramadan, H.E., Porter, D.E., Chen, D.L., Witchel, E.: Improved error reporting for software that uses black-box components. In: Programming Language Des. and Impl (PLDI 2007). ACM Press, New York (2007)
13. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
14. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 575–594. Springer, Heidelberg (2007)
15. Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 97–114. Springer, Heidelberg (2007)
16. Kilian, J.: Founding cryptography on oblivious transfer. In: ACM Symposium on Theory of Comp (STOC 1988), pp. 20–31. ACM Press, New York (1988)

17. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
18. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008)
19. Kruger, L., Jha, S., Goh, E.-J., Boneh, D.: Secure function evaluation with ordered binary decision diagrams. In: ACM CCS 2006, pp. 410–420. ACM Press, New York (2006)
20. Lindell, Y., Pinkas, B.: A proof of Yao's protocol for secure two-party computation. ECCC Report TR04-063, Electronic Colloq. on Comp. Complexity (2004)
21. Lipmaa, H.: Verifiable homomorphic oblivious transfer and private equality test. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, Springer, Heidelberg (2003)
22. Lipmaa, H.: Private branching programs: On communication-efficient cryptocomputing. Cryptology ePrint Archive, Report 2008/107 (2008), `http://eprint.iacr.org/`
23. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay — a secure two-party computation system. In: USENIX (2004), `http://www.cs.huji.ac.il/project/Fairplay`
24. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: ACM-SIAM Symposium On Discrete Algorithms (SODA 2001), pp. 448–457. Society for Industrial and Applied Mathematics (2001)
25. Orlandi, C., Piva, A., Barni, M.: Oblivious neural network computing via homomorphic encryption. European Journal of Information Systems (EURASIP) 2007(1), 1–10 (2007)
26. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
27. Pinkas, B.: Cryptographic techniques for privacy-preserving data mining. SIGKDD Explor. Newsl. 4(2), 12–19 (2002)
28. Piva, A., Caini, M., Bianchi, T., Orlandi, C., Barni, M.: Enhancing privacy in remote data classification. In: New Approaches for Security, Privacy and Trust in Complex Environments, SEC 2008 (2008)
29. Rodriguez, J., Goni, A., Illarramendi, A.: Real-time classification of ECGs on a PDA. IEEE Transact. on Inform. Technology in Biomedicine 9(1), 23–34 (2005)
30. Sadeghi, A.-R., Schneider, T.: Generalized universal circuits for secure evaluation of private functions with application to data classification. In: ICISC 2008. LNCS, vol. 5461, pp. 336–353. Springer, Heidelberg (2008)
31. Sander, T., Young, A., Yung, M.: Non-interactive cryptocomputing for $NC^1$. In: IEEE Symp. on Found. of Comp. Science (FOCS 1999), pp. 554–566. IEEE Computer Society Press, Los Alamitos (1999)
32. Schneider, T.: Practical secure function evaluation. Master's thesis, University of Erlangen-Nuremberg, February 27 (2008)
33. Yao, A.C.: How to generate and exchange secrets. In: IEEE Symposium on Found. of Comp. Science (FOCS 1986), pp. 162–167. IEEE, Los Alamitos (1986)