Combining Signal Processing and Cryptographic Protocol Design for Efficient ECG Classification

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,riccardo.lazzeretti}@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. We describe a privacy-preserving system where a server can classify an ElectroCardioGram (ECG) signal without learning any information about the ECG signal and the client is prevented from gaining knowledge about the classification algorithm used by the server. The system relies on the concept of Linear Branching Programs (LBP) and a recently proposed cryptographic protocol for secure evaluation of private LBPs. We study the trade-off between signal representation accuracy and system complexity both from practical and theoretical perspective. We show how the overall system complexity can be strongly reduced by modifying the original ECG classification algorithm. Two alternatives of the underlying cryptographic protocol are implemented and their corresponding complexities are analyzed to show suitability of our system in real-life applications for current and future security levels.

1 Introduction

Health-care industry is moving faster than ever towards technologies offering personalized online self-service, medical error reduction, customer data collection and more. Such technologies have the potentiality of revolutionizing the way medical data is managed, stored, delivered and ubiquitously made available to millions of users. However, respecting the privacy of customers is a central problem, since privacy concerns may imped the diffusion of new e-health services. In this paper, we consider a scenario for a remote diagnosis service. This service offers the analysis of biomedical signals to provide a preliminary diagnosis and is potentially untrusted. Such a system may either be seen as a stand alone service or as part of a complete e-health system where the service provider in addition to offering a repository of personal medical data, allows to remotely process such

⋆ Supported by the European Union under FP6 project SPEED and by MIUR under project 2007JXH7ET.
⋆⋆ Supported by EU FP6 project SPEED, EU FP7 project CACE and ECRYPT II.

data. In order to preserve the privacy of the users, the server should carry out its task without getting any knowledge about the private data provided by the users. At the same time, the service provider may not be willing to disclose the algorithms it is using to process the signals. In general, one can resort to generic secure two-party computation (2PC) protocols [Yao86,MNPS04] allowing two parties to compute the output of a public function $f(\cdot)$ on their respective private inputs. At the end of the protocol, the only information obtained by the parties is the output of the function $f(\cdot)$, but no additional information about the other party's input. We consider a variant where the function $f(\cdot)$ is a private property of the server. While this can be reduced to secure evaluation of a public function using universal circuits [KS08b], this generic transformation poses an enormous overhead on the protocols. In this work, we consider the privacy-preserving classification of ElectroCardioGram (ECG) signals. Classification of ECG signals has long been studied by the signal processing community, but not yet in the context of a privacy-preserving scenario. In our research we considered the secure implementation of a recently proposed classification algorithm [ASSK07]. The contribution of our research is fourfold. First, we present an efficient protocol for privacy-preserving classification of ECG signals resorting to secure evaluation of Linear Branching Programs (LBP) [BFK$^+$09]. Secure function evaluation with private functions [SYY99,Pin02,KS08b,SS08] 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). Second, we link the representation accuracy of the to-be-processed signals (i.e., the number of bits representing the signals) and hence the complexity of the system, to the classification accuracy. Third, we show how the overall complexity of the system can be reduced by tailoring the ECG classification algorithm to the 2PC scenario. Fourth, we compare two implementations of the secure protocol with respect to different parameter sizes and security levels. The rest of the paper is organized as follows. In §2 the plain version of the ECG classifier is described. In §3 we describe the tool and the notation used in this work. In §4 the LBP concept and the protocols for secure evaluation of private LBPs are summarized. §5 is devoted to the description of the privacy-preserving ECG classification algorithm and the accuracy analysis. Experimental results are discussed in §6 and some conclusions are drawn in §7.

2 ECG Classification in the Plain Domain

In this section, we describe the architecture of the plain domain version of the ECG classifier. In our system we are interested in classifying each heart beat

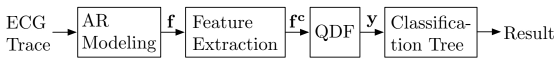

Fig. 1. Block diagram

according to 6 classes: Normal Sinus Rhythm (NSR), Atrial Premature Contractions (APC), Premature Ventricular Contractions (PVC), Ventricular Fibrillation (VF), Ventricular Tachycardia (VT) and SupraVentricular Tachycardia (SVT). The classification algorithm we use is inspired by the work of D. Ge et al. [ASSK07, chapter 8]. The choice of the algorithm is justified first of all by the good classification accuracy it ensures, secondly because it fits well the requirements of a privacy preserving implementation, finally because of its generality. As both, AutoRegressive (AR) models and Quadratic Discriminant Functions (QDF) are often used in automatic medical diagnosis, the protocol described in this paper may represent the basis for a large number of implementations addressing a variety of diverse topics in biomedical signal processing.

The overall architecture of the classifier is summarized by the block diagram in Fig. 1. The input of the system is an ECG chunk corresponding to a single hear beat, that consequently, is classified as an independent entity. For the extraction of heart beats the algorithm proposed in [ASSK07] is used. We assume that the ECG signal is sampled at 250 sample per second and that 300 samples surrounding each peak are fed to the system: 100 samples preceding the beat peak and 200 following it. We also assume that the ECG signal has been pre-filtered by a notch filter removing the noise due to power line interference, electrode contact noise, motion artifact and base line wander [ASSK07].

The ECG classifier considered here relies on a rather general technique based on AR models for ECG description and a subsequent QDF classifier. Specifically, each ECG chunk is modeled by means of a 4-th order AR model. The AR model coefficients can be estimated in several ways; in our system we used a method based upon the Yule-Walker equations [BJR76]. Once the AR model has been computed, five features[4] are extracted, yielding the following vector $\mathbf{f} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, n_e)^T$. The first four features are the coefficients of the AR model and n_e is the number of samples for which the amplitude of the estimation error $|\epsilon_n|$ exceeds a threshold defined as $th = 0.25 \max_n (|\epsilon_n|)$. To perform a QDF classification as a linear operation, the classifier does not operate directly on $\mathbf{f}$. Instead a composite feature vector $\mathbf{f^c}$ is computed containing the features in $\mathbf{f}$, their square values and their cross products, namely:

$$\mathbf{f^c} = (1, f_1, \ldots, f_5, f_1^2, \ldots, f_5^2, f_1 f_2, \ldots f_4 f_5)^T = (f_1^c, \ldots f_{21}^c)^T.$$

The vector $\mathbf{f^c}$ represents the input of the QDF block in Fig. 1. The QDF block projects $\mathbf{f^c}$ onto 6 directions β_i, obtaining a 6-long vector $\mathbf{y}$, that represents the input of the final classification step: $\mathbf{y} = \mathbf{B}\mathbf{f^c}$, where $\mathbf{B}$ is a matrix whose rows are the vectors β_i. The matrix $\mathbf{B}$ contains part of the knowledge embedded

[4] In [ASSK07, chapter 8] 6 features are used, however our experiments have shown that by using 5 features we obtain the same classification accuracy with a lower complexity.

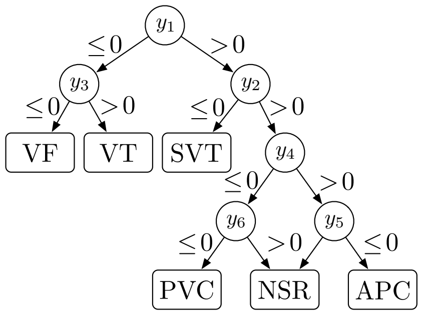

Fig. 2. Binary classification tree for ECG classification

within the classification system, and is computed by relying on a set of training ECGs (see [ASSK07] for the details). For the final classification, the signs of the values y_i are extracted and used to actually classify the ECG, by means of the binary classification tree given in Fig. 2.

3 Preliminaries

In our protocols we combine several standard cryptographic tools (additively homomorphic encryption, oblivious transfer, and garbled circuits) which we summarize in §3.1. Readers familiar with these tools can safely skip §3.1 and continue reading our notational conventions in §3.2.

We denote the symmetric security parameter with t and the asymmetric security parameter with T. Recommended parameters for short-term security are for example $t = 80$ and $T = 1248$ [GQ09].

3.1 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 [Pai99,DJ01] 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 [DJ01].
Parallel Oblivious Transfer (OT). Parallel 1-out-of-2 Oblivious Transfer for m bitstrings of bitlength ℓ, denoted as $\mathsf{OT}m\ell$, is a two-party protocol as shown in Fig. 3. SS inputs m pairs of ℓ-bit strings $S_i = \langle s_i^0, s_i^1 \rangle; i = 1, .., m; s_i^0, s_i^1 \in \{0,1\}^\ell$. $\mathcal{C}$ inputs m choice bits $b_i \in \{0,1\}$. At the end of the protocol, $\mathcal{C}$ learns $s_i^{b_i}$, but nothing about $s_i^{1-b_i}$ whereas SS learns nothing about b_i. We use $\mathsf{OT}m\ell$

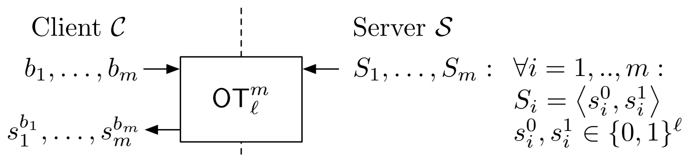

Fig. 3. $\mathsf{OT}m\ell$ - Parallel Oblivious Transfer

as a black-box primitive in our constructions. It can be instantiated efficiently with different protocols [NP01,AIR01,Lip03,IKNP03]. For example the protocol of [NP01] implemented over a suitably chosen elliptic curve has asymptotic communication complexity $m(4t+2r)$ and is secure against malicious $\mathcal{C}$ and semi-honest SS in the random oracle model. The protocol of [AIR01] implemented over a suitably chosen elliptic curve has asymptotic communication complexity $m(12t)$ and is secure against malicious $\mathcal{C}$ and semi-honest SS in the standard model. Extensions of [IKNP03] can be used to reduce the number of computationally expensive public-key operations to be independent of m. We omit the parameters m or ℓ if they are clear from the context.

Garbled Circuit (GC). Yao's Garbled Circuit approach [Yao86], excellently presented in [LP04], 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 SS), 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* $\widetilde{W_i}=\langle\widetilde{w}_i^0,\widetilde{w}_i^1\rangle$ consisting of two secrets, $\widetilde{w}_i^0$ and $\widetilde{w}_i^1$, where $\widetilde{w}_i^j$ is the *garbled value* of W_i's value j. (Note: $\widetilde{w}_i^j$ does not reveal j.) Further, for each gate G_i, SS 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 OT: SS inputs complementary garbled values $\widetilde{W_j}$ into the protocol; $\mathcal{C}$ inputs x_j and obtains $\widetilde{w}_j^{x_j}$ as outputs. Now, $\mathcal{C}$ can evaluate the garbled circuit $\widetilde{C}$ with algorithm 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 [BPSW07] 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 [MNPS04]: 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 π_i is used to select the right table entry for decryption with the key k_i. Extensions of [KS08a] to "free XOR" gates can be used to further improve performance of GC.

3.2 Notation

Number Representation. In the following, a *(signed) ℓ-bit integer* x^ℓ 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^ℓ with $2^\ell \leq N$ is mapped into an element of the plaintext group $\mathbb{Z}_N$ using $m(x^\ell) = \begin{cases} x^\ell, & \text{if } x^\ell \geq 0 \\ N + x^\ell, & \text{if } x^\ell < 0 \end{cases}$.

Homomophic Encryption. $\mathsf{Gen}(1^T)$ denotes the key generation algorithm of the Paillier cryptosystem [Pai99,DJ01] 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 ℓ-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 $\langle \widetilde{w}^0, \widetilde{w}^1 \rangle$ for which we use the corresponding small letters. We group together multiple garbled values to a *garbled ℓ-bit value* $\widetilde{\mathbf{w}}^\ell$ (small, bold letter) which consists of ℓ garbled values $\widetilde{w}_1, \ldots, \widetilde{w}_\ell$. Analogously, a *complementary garbled ℓ-bit value* $\widetilde{\mathbf{W}}^\ell$ (capital, bold letter) consists of ℓ complementary garbled values $\widetilde{W}_1, \ldots, \widetilde{W}_\ell$.

4 Secure Evaluation of Private LBPs

After formally defining Linear Branching Programs (LBP) in §4.1, we present two protocols for secure evaluation of private LBPs. We decompose our protocols into different building blocks similar to the protocol of [BPSW07] and show how to instantiate them more efficiently than in [BPSW07].

The protocols for secure evaluation of private LBPs are executed between a server SS 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 [BPSW07] 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 [BPSW07], 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.

A security analysis of the LBP protocol with detailed proofs in the semi-honest model and extensions to the malicious model together with a discussion on (how to prevent) learning the private LBP with so-called oracle attacks can be found in the full version of [BFK+09], together with a detailed complexity analysis.

4.1 Linear Branching Programs (LBP)

First, we formally define the notion of linear branching programs. We do so by generalizing the BP definition used in [BPSW07]. We note that BPs – and hence also LBPs – generalize binary classification or decision trees and Ordered Binary Decision Diagrams (OBDDs) used in [KJGB06,Sch08].

Definition 1 (Linear Branching Program (LBP)). *Let* $\mathbf{x}^\ell = x_1^\ell, .., x_n^\ell$ *be the* **attribute vector** *of signed* ℓ*-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*,* $1 \leq i \leq d$ *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* $\mathbf{a_i^\ell} = \langle a_{i,1}^\ell, .., a_{i,n}^\ell \rangle$ *is the* **linear combination vector** *consisting of* n *signed* ℓ*-bit integer values and* $t_i^{\ell'}$ *is the signed* ℓ'*-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* $\mathbf{a_i^\ell} \circ \mathbf{x}^\ell \leq t_i^{\ell'}$*;* $\mathit{Right}(i)$ *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 $P_j := \langle c_j \rangle$*,* $d < j \leq z$ *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 $\mathbf{a_1^\ell} \circ \mathbf{x}^\ell \leq t_1^{\ell'}$, move to node $\mathit{Left}(1)$, else to $\mathit{Right}(1)$. 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 ℓ' of the threshold values $t_i^{\ell'}$ has to be chosen according to the maximum value of linear combinations:

$$abs(\mathbf{a_i^\ell} \circ \mathbf{x}^\ell) = abs(\sum_{j=1}^n a_{i,j}^\ell x_j^\ell) \leq \sum_{j=1}^n 2^{2(\ell-1)} = n2^{2(\ell-1)}$$
$$\Rightarrow \ell' = 1 + \lceil \log_2(n2^{2(\ell-1)}) \rceil = 2\ell + \lceil \log_2 n \rceil - 1. \quad (1)$$

As noted above, LBPs can be seen as a generalization of previous representations:
Branching Programs (BP) as used in [BPSW07] are a special case of LBPs. In a BP, in each decision node P_i the α_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 $\ell' = \ell$.

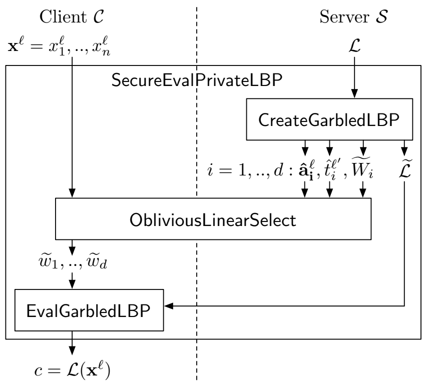
Client $\mathcal{C}$
Server $\mathcal{S}$
$\mathbf{x}^\ell = x_1^\ell, .., x_n^\ell$
$\mathcal{L}$
SecureEvalPrivateLBP
CreateGarbledLBP
$i = 1, .., d : \hat{\mathbf{a}}_i^\ell, \hat{t}_i^{\ell'}, \widetilde{W}_i$
$\widetilde{\mathcal{L}}$
ObliviousLinearSelect
$\widetilde{w}_1, .., \widetilde{w}_d$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

Fig. 4. Secure Evaluation of Private Linear Branching Programs - Structural Overview

Ordered Binary Decision Diagrams (OBDD) as used in [KJGB06,Sch08] are a special case of BPs with bit inputs ($\ell = 1$) and exactly two classification nodes ($P_{z-1} = \langle 0 \rangle$ and $P_z = \langle 1 \rangle$).

4.2 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. 4. The client $\mathcal{C}$ receives an attribute vector $\mathbf{x}^\ell = \{x_1^\ell, \ldots, x_n^\ell\}$ as input, and the server SS 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 SS learns nothing. Of course, both $\mathcal{C}$ and SS wish to keep their inputs private. Protocol SecureEvalPrivateLBP is naturally decomposed into the following three phases (cf. Fig. 4).

CreateGarbledLBP. SS creates a garbled version of the LBP $\mathcal{L}$. This is done similarly to the garbled-circuit-based previous approaches [BPSW07,KJGB06]. 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. 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 SS.

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 is equivalent to Yao's GC evaluation procedure. Here, $\mathcal{C}$ receives the garbled LBP $\widetilde{\mathcal{L}}$ from SS, 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 $c = \mathcal{L}(\mathbf{x}^\ell)$.

4.3 Our Building Blocks

Phase I (offline): CreateGarbledLBP. In this phase, SS generates a garbled version $\widetilde{\mathcal{L}}$ of the private branching program $\mathcal{L}$. Algorithm CreateGarbledLBP [BFK+09] converts the nodes P_i of $\mathcal{L}$ into garbled nodes $\widetilde{P}_{\tilde{i}}$ in $\widetilde{\mathcal{L}}$, as follows. First, we associate a randomly chosen key Δ_i with each node P_i. We use Δ_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 pseudo-random 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 [KJGB06]: Instead of encrypting twice (sequentially, with Δ_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 Δ_i). This eliminates the need for inclusion of classification nodes in the garbled LBP (and necessitates adding a bit denoting the type of the successor's node to the decision node's encryption). This 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 SS in the next phase).

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, $\mathcal{C}$'s input is the private attribute vector $\mathbf{x}^\ell = \{x_1^\ell, \ldots, x_n^\ell\}$ and SS's input are 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 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 $\widetilde{w}_i = \widetilde{w}_i^0$. SS learns nothing about $\mathcal{C}$'s inputs. We note that ObliviousLinearSelect can be viewed as an instance of *conditional oblivious transfer* [BK04]. We give two efficient instantiations for ObliviousLinearSelect in §4.4.

Phase III: EvalGarbledLBP. This algorithm additionally gets the garbled values $\widetilde{w}_1, .., \widetilde{w}_d$ output by ObliviousLinearSelect as inputs and outputs the classification label $c = \mathcal{L}(\mathbf{x}^\ell)$. $\mathcal{C}$ traverses the garbled LBP $\widetilde{\mathcal{L}}$ by decrypting garbled decision nodes along the evaluation path starting at $\widetilde{P}_1$. At each node $\widetilde{P}_{\hat{i}}$,[5] $\mathcal{C}$ 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. It is easy to see that some information 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.

4.4 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 not described here, cause it is an application of the Garbled Circuit. A detailed description may be found in [BFK+09].

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 the hybrid instantiation of ObliviousLinearSelect (see Fig. 5 for an overview), $\mathcal{C}$ generates a key-pair for the additively homomorphic

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

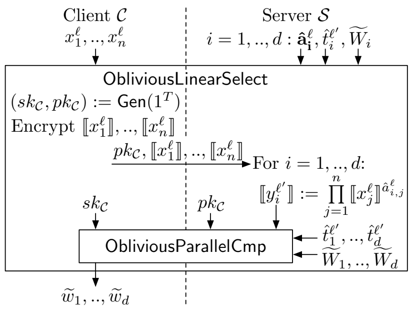
Client $\mathcal{C}$
$x_1^\ell, .., x_n^\ell$
Server $\mathcal{S}$
$i = 1, .., d : \hat{\mathbf{a}}_\mathbf{i}^\ell, \hat{t}_i^{\ell'}, \widetilde{W}_i$
ObliviousLinearSelect
$(sk_\mathcal{C}, pk_\mathcal{C}) := \mathsf{Gen}(1^T)$
Encrypt $[\![x_1^\ell]\!], .., [\![x_n^\ell]\!]$
$pk_\mathcal{C}, [\![x_1^\ell]\!], .., [\![x_n^\ell]\!]$
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. 5. ObliviousLinearSelect - Hybrid

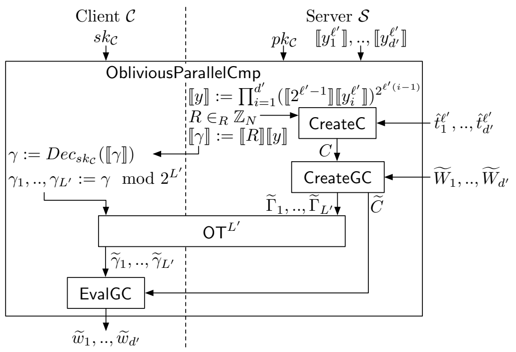
Client $\mathcal{C}$
$sk_\mathcal{C}$
Server $\mathcal{S}$
$pk_\mathcal{C}$
$[\![y_1^{\ell'}]\!], .., [\![y_{d'}^{\ell'}]\!]$
ObliviousParallelCmp
$[\![y]\!] := \prod_{i=1}^{d'}([\![2^{\ell'-1}]\!][\![y_i^{\ell'}]\!])^{2^{\ell'(i-1)}}$
$R \in_R \mathbb{Z}_N$
$[\![\gamma]\!] := [\![R]\!][\![y]\!]$
CreateC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

Fig. 6. ObliviousParallelCmp

encryption scheme and sends the public key $pk_\mathcal{C}$ together with the homomorphically encrypted attributes $[\![x_1^\ell]\!], .., [\![x_n^\ell]\!]$ to SS. Using the additively homomorphic property, SS can compute the linear combination of these ciphertexts with the private coefficients $\hat{\mathbf{a}}_\mathbf{i}^\ell$ as $[\![y_i^{\ell'}]\!] := [\![\sum_{j=1}^n \hat{a}_{i,j}^\ell x_j^\ell]\!] = \prod_{j=1}^n [\![x_j^\ell]\!]^{\hat{a}_{i,j}^\ell}$, $1 \leq i \leq d$. 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 $y_i^{\ell'} \leq \hat{t}_i^{\ell'}$ and $\widetilde{w}_i^1$ otherwise. ObliviousParallelCmp ensures that neither $\mathcal{C}$ nor SS 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. 6).* The idea that is at the basis of the ObliviousParallelCmp protocol is that SS blinds the encrypted value $[\![y_i^{\ell'}]\!]$ in order to hide the encrypted plaintext from $\mathcal{C}$. To achieve this, SS adds a randomly chosen value $R \in_R \mathbb{Z}_N$ [6] 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 [BPSW07] with the following technical trick:

Packing. Usually, the plaintext space of the homomorphic encryption scheme $\mathbb{Z}_N$ is substantially larger than the encrypted values $y_i^{\ell'}$. Hence, multiple encryptions, say d', can be packed together into one ciphertext before blinding and sending it to $\mathcal{C}$. This reduces the communication complexity and the number of decryptions that need to be performed by $\mathcal{C}$ by a factor of d'. For this, the encrypted values $-2^{\ell'-1} < y_i^{\ell'} < 2^{\ell'-1}$ are shifted into the positive range $(0, 2^{\ell'})$ first by adding $2^{\ell'-1}$ and afterwards are concatenated by computing $[\![y]\!] = [\![\sum_{i=1}^{d'} 2^{\ell'(i-1)}(y_i^{\ell'} +$

[6] In contrast to [BPSW07], we choose R from the full plaintext space in order to protect against malicious behavior of $\mathcal{C}$.

$2^{\ell'-1})\rrbracket = \prod_{i=1}^{d'}(\llbracket 2^{\ell'-1}\rrbracket \llbracket y_i^{\ell'}\rrbracket)^{2^{\ell'(i-1)}}$. The packed ciphertext $\llbracket y \rrbracket$ encrypts a $L' = d'\ell'$ bit value now.

4.5 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 [KJGB06], private OBDDs [Sch08], and private BPs [BPSW07] 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 [KJGB06,Sch08,BPSW07] when instantiated for the respective special case functionality.

In the following, we summarize the employed techniques. We stress that our improvements, achieved by combining our new technical tricks and some previously known techniques, are at the conceptual protocol level and will translate directly into any implementation, i.e., are independent of implementation details such as programming languages or hardware used.

Incorporate classification nodes into decision nodes: In contrast to previous protocols given in [KJGB06,Sch08,BPSW07], the size of our improved method for constructing garbled LBPs depends on the number of decision nodes d only which is smaller than the total number of nodes z, a full classification tree has $z = 2d + 1$ nodes. Hence, $\mathcal{C}$ cannot infer the number of classification nodes in our protocols. As additionally the number of attributes for which the ObliviousLinearSelect protocol needs to be run is reduced from z down to d, the communication complexity of our hybrid protocol — except sending the encrypted attributes (HE) — grows linearly in d instead of z. For full classification trees this results in an overall improvement by a factor of two. **Tiny LBPs:** For tiny LBPs with $d \leq 10$ decision nodes, e.g., the privacy-preserving remote diagnostics examples for iptables and mpg321 in [BPSW07], or the medical ECG classification application described in §2, our alternative method for garbled LBPs results in a substantially smaller communication complexity than existing solutions. The garbled LBP can be constructed using a single Yao gate with d inputs as described next. The garbled LBP $\widetilde{\mathcal{L}}$ needs to obliviously map the garbled inputs $\widetilde{w}_1, .., \widetilde{w}_d$ to the corresponding classification label c as explained in §4.2. This can trivially be implemented with a Yao gate with d inputs which encrypts for each of the 2^d possible input combinations the index of the corresponding label. As the total number of classification nodes is $z - d$, their index can be encoded with $\lceil \log(z - d) \rceil$ bits. As the size of this alternative construction for garbled LBPs grows exponentially in d, this is feasible for tiny LBPs only. While – in contrast to the method described in §4.3 – this method reveals the number of classification nodes and their labels but hides the length of the evaluation path without need for padding with dummy decision nodes.

Point-and-permute: The protocols of [KJGB06,BPSW07] use the semantically secure encryption scheme with 'special' properties of [LP04] where trial-decryptions of all entries must be performed until the 'right' entry was found (which can be decided using the special properties). To achieve the special properties, the ciphertext size is increased by κ (the statistical correctness parameter) additional bits [LP04]. In contrast to this, our protocols use a point-and-permute technique as described in §3.1 to directly identify the right entry to decrypt during evaluation of the GC as well as for the garbled LBP. This reduces the computation complexity for $\mathcal{C}$ as no trial-decryptions need to be performed. Also a standard semantically secure encryption scheme without padding can be used in our protocols which results in reduced communication complexity.

Key-offsets: In the protocol of [BPSW07], the garbled circuits are encrypted symmetrically with a node-specific key. During evaluation, $\mathcal{C}$ obtains this key, decrypts the GC, and evaluates it. In contrast to this, our protocols use node-specific key-offsets Δ_i as described in §4.3 which result in lower computation complexity for both, SS and $\mathcal{C}$ (adapted from [KJGB06]).

Packing: The hybrid ObliviousLinearSelect protocol improves over the protocol of [BPSW07] by packing together multiple ciphertexts before sending them back to $\mathcal{C}$ as described in §4.4. This reduces the communication complexity and the number of public-key decryptions that $\mathcal{C}$ needs to perform by a factor of $d' = \frac{T-\kappa}{\ell'}$

5 Privacy-Preserving ECG Classification

In this section, we describe how LBP may be used for privacy-preserving classification of ECG signals. Variables names used in this section are the same used in [ASSK07, chapter 8] and §2, different from that used in the section §4, where more generic names are prefered.

Before describing the privacy-preserving ECG classification protocol, we define the players of the protocol and the data that needs to be protected. A first requirement is that the server SS, who is running the classification algorithm on client's ECG signal, learns neither any information about the ECG signal nor the final result of the classification. At the same time, the client $\mathcal{C}$ should not get any information about the algorithm used by SS, except for the output of the classification. The latter point deserves some explanation. We assume that the general form of the classifier used by SS is known, however the parameters of the classifier need to be kept secret. By referring to the description given in §2, the algorithm parameters that SS aims at keeping secret are the matrix $\mathbf{B}$ and the classification tree of Fig. 2. This is a reasonable assumption since the domain specific knowledge needed to classify the ECGs and the knowledge got from the training, a knowledge that SS may want to protect, resides in the classification tree and the matrix $\mathbf{B}$.

In order to introduce the privacy-preserving ECG classification protocol, we observe that the classification algorithm described in §2 is nothing but an LBP with $\mathbf{f}^{\mathbf{c},\ell}$ as attribute vector, and 6 nodes $P_i = \left\langle \beta_{\mathbf{i}}^{\ell}, 0 \right\rangle, i = 1, .., 6$, where $\mathbf{f}^{\mathbf{c},\ell}$ and $\beta_{\mathbf{i}}^{\ell}$ are ℓ-bit representations of the feature and projection vectors. In this way, the general scheme for the privacy-preserving implementation of the classifier assumes the form given in Fig. 7. All steps until the computation of the composite

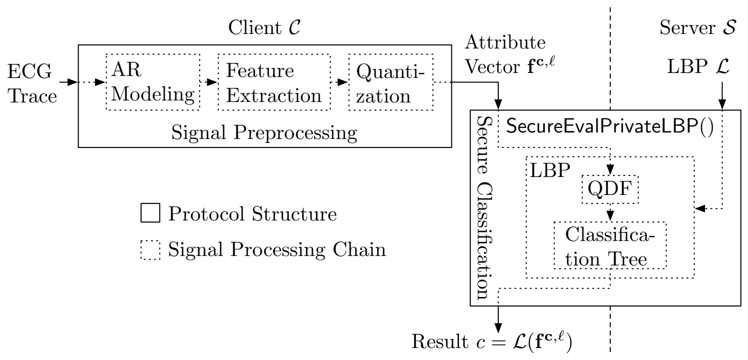

Fig. 7. Privacy-preserving ECG diagnosis

feature vector are performed by $\mathcal{C}$ on the plain data. Such a choice does not compromise the security of the system from the server's point of view, since SS is not interested in keeping the structure of the classifier secret, but only in preventing users from knowing the matrix $\mathbf{B}$ and the classification tree. On the contrary, all the steps from the projection onto the directions β_i's, through the final classification are carried out securely. Note that with respect to the overall architecture depicted in Fig. 1, we added a quantization step before the encryption of the composite feature vector. The need for such a step comes from the observation that the parameters $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ resulting from the AR model estimation procedure are usually represented as floating point numbers, a representation that is not suitable for 2PC protocols which compute on numbers represented as integers only. For this reason the elements of the composite feature vector $\mathbf{f}^{\mathbf{c}}$ are quantized and represented in integer arithmetic for subsequent processing[7]. Note that the choice of the quantization step, and consequently the number of bits used to represent the data (ℓ in the LBP terminology), is crucial since on one side it determines the complexity of the overall secure protocol and on the other side it has an impact on the accuracy of the ECG classification.

5.1 Quantization Error Analysis

In this section we estimate the impact that the quantization error introduced passing from $\mathbf{f}^{\mathbf{c}}$ to $\mathbf{f}^{\mathbf{c},\ell}$ and from $\beta_{\mathbf{i}}$ to $\beta_{\mathbf{i}}^{\ell}$ has on the classification accuracy. Such an analysis will be used to determine the minimum number of bits (ℓ) needed to represent the attribute vector and the linear combination vectors of the LBP. The value of ℓ influences the complexity of the secure classification protocol for two main reasons. As already outlined in §4, the main ingredients of the protocols for secure evaluation of private LBPs are garbled circuits and additively homomorphic encryption. In the case of garbled circuits, the input of the protocol are the single bits used to represent $\mathbf{f}^{\mathbf{c},\ell}$ and $\beta_{\mathbf{i}}^{\ell}$. It is obvious, then, that the greater the number of bits, the more complex the resulting protocol will be. With regard

[7] In the same way the coefficients of matrix B, representing the combination vectors of the LBP, are represented as integer numbers.

to computing on homomorphically encrypted data, we observe that after each multiplication carried out in the encrypted domain, the number of bits necessary to represent the output of the multiplication increases (it approximately doubles)[8]. Since it is not possible to carry out truncations in the encrypted domain, it is necessary that the size of the ring used by the homomorphic cryptosystem is large enough to contain the output of the computations without an overflow which would cause an invalid result. Augmenting the number of bits used to represent the inputs of the LBP may require to increase the size of the needed cryptosystem ring which results in an increased protocol complexity.

To start with, we observe that quantization is applied to the composite feature vector $\mathbf{f^c}$, that is used to compute the vector $\mathbf{y}$, through multiplication with the matrix $\mathbf{B}$. After such a step, only the signs of vector $\mathbf{y}$ are retained, hence it is sufficient to analyze the effect of quantization until the computation of the sign of $\mathbf{y}$. As to the processing steps carried out by the client prior to quantization, we assume that all the blocks until QDF are carried out by using a standard double precision floating point arithmetic. In order to simplify the notation, we consider only the computation of one coefficient of the vector $\mathbf{y}$. The function to be computed is a simple inner product: $y = \sum_j \beta_j f_j^c$ where the index i has been omitted, and β_j and f_j^c are real numbers. The quantized version of the above relationship can be expressed as follows:

$$\begin{aligned} \beta_{q,j} &= \rho_1 \beta_j + \varepsilon_{1,j} = \lfloor \rho_1 \beta_j \rceil \\ f_{q,j}^c &= \rho_2 f_j^c + \varepsilon_{2,j} = \lfloor \rho_2 f_j^c \rceil \end{aligned} \tag{2}$$

where ρ_1 and ρ_2 are positive integers and $\varepsilon_{1,j}$ and $\varepsilon_{2,j}$ are the quantization errors affecting $\beta_{q,j}$ and $f_{q,j}^c$ respectively. By using the above relations it is possible to evaluate the final error due to quantization:

$$\begin{aligned} &\sum_{j=0}^{N-1} \left(\rho_1 \beta_j + \varepsilon_{1,j} \right) \left(\rho_2 f_j^c + \varepsilon_{2,j} \right) = \\ &\qquad = \rho_1 \rho_2 \Bigg(y + \underbrace{\sum_{j=0}^{N-1} \frac{\beta_j \varepsilon_{2,j}}{\rho_2} + \sum_{j=0}^{N-1} \frac{f_j^c \varepsilon_{1,j}}{\rho_1} + \sum_{j=0}^{N-1} \frac{\varepsilon_{1,j} \varepsilon_{2,j}}{\rho_1 \rho_2}}_{\varepsilon} \Bigg) \end{aligned} \tag{3}$$

where ε indicates the error on the scalar product once the scaling factor $\rho_1 \rho_2$ is canceled out. By letting $\max(|\beta_j|) = M_b$, $\max(|f_j^c|) = M_f$ and by noting that $\max(|\varepsilon_{1,j}|) = \max(|\varepsilon_{2,j}|) = \frac{1}{2}$, we have:

$$\varepsilon \leq \frac{N}{2\rho_1 \rho_2} \left(\rho_1 M_b + \rho_2 M_f + \frac{1}{2} \right) \leq \varepsilon^* \tag{4}$$

where ε^* is a target maximum error that we do not want to exceed. Given ε^*, choosing the optimum values of ρ_1 and ρ_2 is equivalent to a constrained

[8] The same observation holds for additions, however additions have a negligible effect with respect to multiplications.

minimization problem in which the function to be minimized is $\rho_1\rho_2$ (since this is equivalent to minimize the number of bits necessary to represent the output of the scalar product) and the constraint corresponds to equation (4), that is:

$$\rho_1 \geq \frac{N(2\rho_2 M_f + 1)}{4\rho_2\varepsilon^* - 2NM_b}. \tag{5}$$

To ensure that ρ_1 is a positive integer, we require $2\rho_2\varepsilon^* - NM_b > 0$, yielding the following minimization problem:

$$\min_{\rho_2 > \frac{NM_b}{2\varepsilon^*}} \rho_2 \frac{N(2\rho_2 M_f + 1)}{4\rho_2\varepsilon^* - 2NM_b}. \tag{6}$$

By solving (6) we obtain the solutions:

$$\rho_2 = \frac{1}{2M_f\varepsilon^*}\left(NM_bM_f + \sqrt{NM_bM_f(\varepsilon^* + NM_bM_f)}\right), \tag{7}$$

$$\rho_1 = \frac{1}{2M_b\varepsilon^*}\left(NM_bM_f + \sqrt{NM_bM_f(\varepsilon^* + NM_bM_f)}\right). \tag{8}$$

5.2 Speeding up the System

By referring to the analysis in the previous section, we must consider that in our case $N = 21$, however the values of M_b and M_f are not known. In fact, the coefficients of the AR model and matrix B are not bounded. However, considering that in practical applications AR model coefficients are rather small (lower than 10 for ECG signals) and observing that the 5-th component of the feature vector $\mathbf{f}$ can be at most 300, hence in $\mathbf{f^c}$ we surely have a component that is at most $9 \cdot 10^4$. We may then let M_f to be the closest power of 10, i.e., $M_f = 10^5$. At the same time, in our experiments we never observed a matrix B with coefficients larger than $M_b = 10^5$. Finally by examining the data of the ECG MIT Database[9] we found that $\varepsilon^* = 10^{-5}$ ensures a sufficient classification accuracy. By using these settings the bit size of the values in input turned out to be 56 bit. As to the ring size for homomorphic encryption we found that the ring size imposed by security standards, e.g., 1248 bits and more [GQ09], is always sufficient to accommodate all the intermediate and final results of the computation.

The analysis reported above is mainly based on worst case assumptions. In practice, we may expect that the number of bits necessary for a good classification accuracy is lower than 56. To investigate this aspect, we implemented a simulator to exactly understand which is the minimum number of bits that can be used. The results we obtained by running the simulator on the MIT Database of ECG signals are shown in Fig. 8. This figure shows the accuracy of the system as a function of ℓ. As we can see $\ell = 44$ is sufficient to guarantee the same performance of a non-quantized implementation.

[9] `http://www.physionet.org/physiobank/`

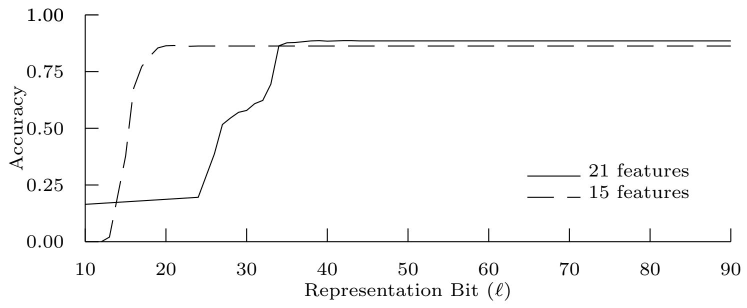

Fig. 8. Classification accuracy of dataset using 21 and 15 features

In order to further speed up the system, we tested a version of the ECG classifier with a reduced number of features. Specifically, we reduced $\mathbf{f}$ by eliminating the feature n_e. In this way, we obtain a 15-coefficient $\mathbf{f^c}$. Obviously the reduction of the feature space gives also a reduction of the accuracy, but this reduction is quite negligible: our experiments, in fact, indicate that the accuracy decreases only from 88.57% to 86.30%. On the other hand, as it will be shown in the next section, by removing one feature we gain a lot from a complexity point of view. Such a gain is already visible in Fig. 8, where we can see that with the reduced set of features a value of ℓ as low as 24 is enough to obtain the same performance of a non-quantized version of the classifier.

6 Complexity Analysis

To evaluate the communication and computation complexity of the Hybrid and the GC protocols, we implemented both protocols in C++ using the Miracl library[10]. The following tests were run on two PCs with 3 GHz Intel Core Duo processor and 4GB memory connected via Gigabit Ethernet. The security parameters in the protocols of [BFK+09] are denoted by T for the bitlength of the RSA modulus for Paillier encryption [Pai99] in the Hybrid protocol, and t for the symmetric security parameter which determines the performance of the GC protocol using an elliptic-curve based oblivious transfer protocol. In our implementation, we chose these security parameters according to common recommendations [GQ09] as $T = 1248, t = 80$ for short-term security (recommended use up to 2010) and $T = 2432, t = 112$ for medium-term security (up to 2030). We measured the complexity of both protocol instantiations for the parameter sizes proposed in §5.2:

In test #1, we represent the features of $\mathbf{f}^{\mathbf{c},\ell}$ with $\ell = 56$ bits, as obtained from the theoretical estimations.

In test #2, the features are represented with $\ell = 44$ bits, the lower value obtained from the practical tests.

[10] `http://www.shamus.ie`

In test #3, we measure how the optimizations of §5.2 increase the efficiency of the protocols. While test-cases #1 and #2 were run for short-term security parameters only, in this test case we consider short-term (#3) and medium-term (#3*) security.

#	Fea-tures	N	ℓ	Protocol Type	Communication Client [kBytes]		Computation Client [s]		Computation Server [s]	
					sent	received	cpu	total	cpu	total
1	5	21	56	Hybrid	20.7	119.1	2.3	35.4	5.4	34.2
				GC	24.1	67435.6	7.2	64.5	17.3	64.7
2	5	21	44	Hybrid	17.7	94.5	2.0	29.0	4.8	27.6
				GC	19.0	41573.6	4.7	48.5	11.5	48.8
3	4	15	24	Hybrid	10.9	52.4	1.3	18.7	3.3	16.2
				GC	7.4	8788.4	1.3	17.5	3.1	19.2
3*	4	15	24	Hybrid	17.6	71.7	6.5	40.5	16.3	30.9
				GC	10.2	11984.3	3.0	20.4	4.6	20.8

* medium-term security

Table 1. Performance of protocols for secure ECG classification

Table 1 shows the results obtained from running these tests. Specifically, the table contains the communication complexity (separated into data sent and received by the client) and the computation complexity for the client and the server (separated into CPU time and total time which additionally includes data transfer and idle times). From these measurements we draw the following conclusions:

a) Parameter Sizes: The performance of both protocols in test #2 is slightly better than that of test #1 due to smaller size of ℓ. Reducing the number of features in test #3 results in substantially improved protocols while the classification accuracy is only negligibly decreased as discussed in §5.2.

b) Communication Complexity: While the data sent by the client is approximately the same for both protocols (few kBytes), the received data in the GC protocol (MBytes) is by an order of magnitude larger than in the Hybrid protocol (kBytes). However, this asymmetric communication complexity of the GC protocol matches today's asymmetric network connections (e.g., ADSL or mobile networks), where the upstream is limited, while tens of MBytes can be downloaded easily. Future research should concentrate on further reducing the communication complexity of GC.

c) Computation Complexity (short-term security): For the test cases #1 and #2 the computation complexity of the Hybrid protocol is better by a factor of three in CPU time and factor two in total time, whereas for the optimized test case #3 both protocols have approximately the same computation complexity. Hence, for short-term security, the Hybrid protocol is better than the GC protocol with respect to computation and communication complexity (see also 'b)' above).

d) Computation Complexity (medium-term security): Increasing the security parameters has a much more dramatic effect on the computation complexity of the Hybrid protocol than on that of the GC protocol (see test #3 vs. #3*). This effect results from the asymmetric security parameter T being almost doubled, whereas the symmetric security parameter t is only slightly increased. We stress that this loss in performance of additively homomorphic encryption for realistic security parameter sizes is often neglected in literature or hidden by choosing relatively small moduli sizes of $T = 1024$ bit. For medium-term security, the GC protocol is substantially better than the Hybrid protocol besides the amount of data received by the client (see discussion in 'b)' above).

7 Conclusions

Privacy-preserving processing of medical signals calls for the application of cryptographic two-party computation techniques to medical signals. While in principle this is always possible, the development of efficient schemes that minimize the computation and communication complexity is not trivial, since it requires a joint design of the signal processing (SP) and cryptographic aspects of the system. In this paper we have presented an efficient and secure system for privacy-preserving classification of ECG signals based on a recently proposed 2PC protocol and a careful design of the SP algorithm used to classify the ECG. In particular, the optimization of the SP part substantially improved the performance of the secure protocols. We experimentally compared two different implementations of the system, one relying on garbled circuits (GC) and one on a hybrid combination of the homomorphic Paillier cryptosystem and GCs (Hybrid). While from communication complexity perspective the Hybrid protocol is clearly better, the computation complexity of both protocol is similar for short-term security parameters, whereas for medium-term security the GC based protocol is preferable.

References

AIR01. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology – EUROCRYPT'01*, volume 2045 of *LNCS*, pages 119–135. Springer, 2001.

ASSK07. U. R. Acharya, J. Suri, J. A. E. Spaan, and S. M. Krishnan. *Advances in Cardiac Signal Processing.* Springer, 2007.

BFK$^+$09. M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. Secure evaluation of private linear branching programs with medical applications. In *14th European Symposium on Research in Computer Security (ESORICS'09)*, LNCS. Springer, 2009. Full version available at `http://eprint.iacr.org/2009/195`.

BJR76. G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time series analysis.* Holden-day San Francisco, 1976.

BK04. I. F. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Advances in Cryptology – ASIACRYPT'04*, volume 3329 of *LNCS*, pages 515–529. Springer, 2004.

BPSW07. J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *ACM Conference on Computer and Communications Security (CCS'07)*, pages 498–507. ACM, 2007.

DJ01. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public-Key Cryptography (PKC'01)*, LNCS, pages 119–136. Springer, 2001.

GQ09. D. Giry and J.-J. Quisquater. Cryptographic key length recommendation, March 2009. `http://keylength.com`.

IKNP03. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO'03*, volume 2729 of *LNCS*. Springer, 2003.

KJGB06. L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *ACM Conference on Computer and Communications Security (CCS'06)*, pages 410–420. ACM Press, 2006.

KS08a. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

KS08b. V. Kolesnikov and T. Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography and Data Security (FC'08)*, volume 5143 of *LNCS*, pages 83–97. Springer, 2008.

Lip03. H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Advances in Cryptology – ASIACRYPT'03*, volume 2894 of *LNCS*. Springer, 2003.

LP04. Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. ECCC Report TR04-063, Electronic Colloq. on Comp. Complexity, 2004.

MNPS04. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX*, 2004. `http://www.cs.huji.ac.il/project/Fairplay`.

NP01. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium On Discrete Algorithms (SODA'01)*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

Pin02. B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *SIGKDD Explor. Newsl.*, 4(2):12–19, 2002.

Sch08. T. Schneider. Practical secure function evaluation. Master's thesis, University of Erlangen-Nuremberg, February 27, 2008.

SS08. A.-R. Sadeghi and T. Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *International Conference on Information Security and Cryptology (ICISC'08)*, volume 5461 of *LNCS*, pages 336–353. Springer, 2008.

SYY99. T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *IEEE Symp. on Found. of Comp. Science (FOCS'99)*, pages 554–566. IEEE, 1999.

Yao86. A. C. Yao. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.