Private Client-side Profiling with Random Forests and Hidden Markov Models

George Danezis[1], Markulf Kohlweiss[1], Benjamin Livshits[1], and Alfredo Rial[2]

[1]Microsoft Research
{gdane, markulf, livshits}@microsoft.com

[2]IBBT and KU Leuven, ESAT-COSIC, Belgium
alfredo.rial@esat.kuleuven.be

Abstract. Nowadays, service providers gather fine-grained data about users to deliver personalized services, for example, through the use of third-party cookies or social network profiles. This poses a threat both to privacy, since the amount of information obtained is excessive for the purpose of customization, and authenticity, because those methods employed to gather data can be blocked and fooled.
In this paper we propose privacy-preserving profiling techniques, in which users perform the profiling task locally, reveal to service providers the result and prove its correctness. We address how our approach applies to tasks of both classification and pattern recognition. For the former, we describe client-side profiling based on random forests, where users, based on certified input data representing their activity, resolve a random forest and reveal the classification result to service providers. For the latter, we show how to match a stream of user activity to a regular expression, or how to assign it a probability using a hidden Markov model. Our techniques, based on the use of zero-knowledge proofs, can be composed with other protocols as part of the certification of a larger computation.

1 Introduction

Many popular business models rely on profiling users to deliver customised services. Currently, privacy-sensitive fine-grained data about each user's activities is gathered on the server side to perform this personalization. As an alternative strategy, we propose a set of techniques that allow for user profiling to be performed in a privacy-preserving manner.

By way of motivation, consider a retailer of home heating equipment that wishes to tailor prices or rebates according to a household's occupancy patterns. Similarly, a soft drink manufacturer may wish to provide free drink samples to loyal customers of its competitors. More generally, a marketing company may wish to classify customers according to their lifestyle to better target reduction coupons. A news site may adjust its content according to user preferences on a social network. These are all examples of *personalization*, a strategy that necessitates knowing more about the user, or user *profiling*.

In recent years, several mechanisms have emerged for discerning user preferences on a large scale. The most notable of them is third-party advertising, which involves partially observing user browsing history through third-party cookies to understand user preferences. Similarly, social network platforms allow third-party apps to query user profiles and "likes" for personalization purposes. If this information were only to be used for the purposes of personalization, in both settings the amount of detailed information relating to the user would seem to be excessive. To summarize, the current approach has the following drawbacks:

- **Privacy:** service providers not only learn the profiles of customers, which might in themselves be privacy-sensitive, but also acquire a mass of fine-grained data about users, including click streams, previous purchases, social contacts, etc.
- **Authenticity:** the correctness of profiling is doubtful as the data has been gathered in an ad-hoc manner, often through third party networks or web "bugs" or beacons, which can be blocked and fooled. Moreover, a determined user may interfere with cookie-based profile gathering by geting involved in so-called cookie swaps — rings of people exchanging tracking cookies to confuse the trackers.

In this work, we propose techniques that perform profiling on the client side using state-of-the-art machine learning procedures, namely random forests [11]. Clients can conclusively prove that they have performed the classification task correctly, with reference to certified data representing user activity. These certificates serve as a *root of trust*: a service provider can unreservedly rely on the computed profile for business decisions, such as giving free samples, coupons or rebates, without learning anything more than the decision or fearing that the user is cheating.

In addition to resolving random forests, we show how to match a stream of user activity against a regular expression. If desired, we can also assign a probability to a stream of user activity using a hidden Markov model [4]. This has applications in matching against streams of user activity, such as finding patterns in bank transactions, for example. The same guarantees relating to privacy and authenticity hold in both of these cases.

Previous private data mining techniques are too specific to the problem they solve and can only be used in isolation [44, 36]. In contrast, the techniques presented in this paper are *fully composable* with other zero-knowledge protocols. As a result, they may be used as part of certifying a larger computation.

2 System Overview

At a high level, we present cryptographic mechanisms that allow a user to prove some certified features match a class or a sequence, defined by a random forest, regular expression, or a hidden Markov model, respectively.

Our protocol takes place between a prover, a verifier, and two authorities: A and A'. The basic interactions between those entities are illustrated in Figure 1

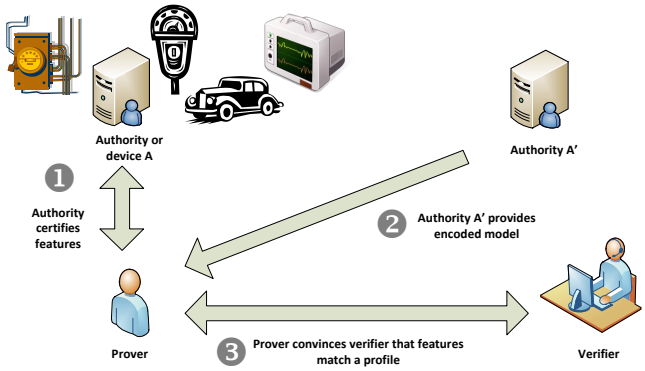

Fig. 1. An overview of the key interactions between all parties in the system.

and described in detail below. We assume that there is an implicit setup phase, during which all principals in the system generate their public/private key pairs and distribute them securely to all other entities. The prover and the verifier also generate signature and verification keys for purposes of authentication.

For the purposes of the protocol, a prover must be provided with a set of certified features (Figure 1, step 1). Abstractly, these features are signed by an authority A that is trusted by the verifier to authoritatively associate features with the prover. In practice, this authority could be a third party service, such as a trusted social network platform that observes user actions and provides a signed account of them for third parties to use. Or it could be a trusted hardware device, such as a smart meter that is installed on user premises to provide certified readings. Similarly an automotive on-board unit could provide trusted readings, such as time and vehicle speed, that need to be matched or classified for the purposes of billing for pay-per-mile insurance. Finally, the user could self-certify some features, and even keep them secret, and simply attest them with a third party using a blind signing protocol.. The features need to be encoded in a specific manner as described in Section 5 and Section 6.

A separate entity A', that could be the verifier, cryptographically encodes a decision forest or a hidden Markov model and provides it to the prover (Figure 1, step 2). In Section 5 and Section 6 we describe how to encode random forest and hidden Markov model classifiers to support efficient proof of their operation. In theory, the classifiers themselves could be kept secret from the verifier, since the verification does not require them. This may enable some business models, but is fragile due to the fact that the prover has to know the full details of the classifiers to construct the necessary proofs.

Finally, the prover and verifier interact, in such a way that the verifier is provided with the outcome of the random forest classification or hidden Markov model match probability, along with a proof that it was performed correctly

(Figure 1, step 3). No other information about the features is revealed. The subsequent sections of this paper present the details of these proofs. Standard techniques can be used to turn the interactive protocol into a non-interactive one. In that setting, the prover simply constructs the proof and sends it to the verifier that can either accept or reject it.

Our protocols protect privacy in the sense that they leak no more information about the detailed features than what becomes known through the decision of a classifier or a match with a hidden Markov chain. We note that such classifiers could be build to leak specific features, and therefore a user needs to know what being subjected to such profiling entails. Since the classifier is public it can be executed, or not, according to the privacy preference of the prover.

The interactions between the authority certifying the features, the prover and the verifier are reminiscent of anonymous credential protocols. The key difference of our approach from such protocols is that we do not attempt to hide the identity of the prover, but rely on hiding the detailed user features to preserve privacy. The protocols can be extended with relative ease to protect the identity of the user but this goes beyond the scope of this work.

3 Applications

This section outlines some of the applications of the theory presented in this paper. The application focus for this paper is on end-user settings, many of which pertain to the context of everyday life, rather than enterprise or business-to-business scenarios.

Behavioural advertising: Perhaps, the most common application setting on people's minds today is that of behavioural profiling – web users are classified based on their pattern of visiting web pages, according to either a taxonomy or by associating keywords with their profile, typically, for more accurate ad targeting. Our techniques allow the user to run the classification algorithm *locally* using fine-grained behavioural information — usually, on their URL stream — and only reveal their aggregate profile to an advertiser or web site for the purposes of customization or ad targeting. Our schemes could make use of features certified by social networking sites, or even certified cookies gathered by browsing on the internet. As has been proposed in the literature [40, 30, 43, 26], in this case, a set of ads can be matched against the user profile without revealing precise user web browsing history.

P2P dating & matchmaking: A common functionality of on-line social sites is the ability to filter users based on one or more criteria. More specifically, imagine a P2P dating site. The site is decentralized, i.e. P2P in order to avoid sharing rather personal details with the site owner. The process of matchmaking, i.e. filtering of the users with respect to criteria such as music tastes or hair color, would be done locally, on each of the P2P nodes. Of course, we need to make sure that this local filter is not compromised to always cheerfully, but deceitfully answer "yes, I am a match." We can prevent this by requiring users to commit to

their preferences and attributes at most once, and then use the same, committed, profile when answering matching queries from different parties.

In this scenario, trust is rooted in the fact that users cannot have their attributes certified at will, and can only do so in a restricted manner: either through paying some money to get their subscription to the dating site to establish an identity, or any other Sybil defence mechanism [35].

Financial logs: In banking applications, it is common to look for patterns over user transaction data. One common application is fraud detection. For example, two or more bank charges in a stream of account transactions per month or too many charges by a particular merchant may be indicative of fraud. Banks themselves are often interested in flagging certain kinds of transactions, primarily for fraud prevention reasons. Governments are keen to oversee transactions to avoid tax-evasion or money laundering.

A more specific application using our techniques is a lender who is interested in querying your transaction history for determining if you are a reliable borrower, which might include checking for more than two late fees within a month. Our techniques could use transaction financial logs provided by financial institutions or shops to prove certain properties to third parties, such as a lending institution.

Insurance: Metered automotive insurance is an example of an application where revealing too much about the user's driving habits can be privacy-compromising, as it will leak excessive data about their whereabouts. However, certain behavioural patterns on the part of the user can be utilized to flag dangerous or reckless behaviour, which are legitimate to inquire about for the purpose of providing and charging for automotive insurance.

For instance, a driver that alternates prolonged day and night driving within the same 24- or 48-hour period could be at risk for an accident. Someone who starts the car more than, say, ten times a day does a fair bit of city driving. Such patterns may be mined to determine the right insurance premium. To support such settings, a trusted on-board unit in the car can provide certified readings of speed and time that are mined to detect the presence of certain driving patterns, without revealing the rest.

Bio-medical & genetic: A number of services have been proposed on the basis of individual genetic or biological profiles. These can include claiming a state benefit due to some debilitating genetic condition or even testing a genetic fingerprint as part of a criminal investigation. Our techniques could be used to support privacy in such settings: an authority could provide a user with a certified generic profile as detailed as their DNA profile. Third parties can then provide certified models that match specific genes or other biological profiles to tune their services. The results computed by these third parties can be shared, yet the detailed biological profile will never be revealed.

3.1 Related Work

Random forests are a very popular classification algorithm first proposed by Breiman [11]. Their performance, efficiency and potential for parallelization has made random forests popular for image classification [8], and they have been widely used in bio-informatics [24]. Hidden Markov models [31] have found applications in speech recognition [39] as well as bio-informatics [32]. The fact that both techniques have serious medical applications motivates us further to perform classification and matching tasks without revealing raw data.

Previous works have considered privacy in the context of classification tasks. Pinkas [38] argues that arbitrary multi-party computation can be used to jointly perform privacy-preserving data mining when different parties have secrets. The key building block of the presented protocol is oblivious transfer, which requires interactions among parties. Our techniques differ in that the only party with secrets is the user, and all other parties only care about the integrity and authenticity of the classification procedure. As a result, our protocols are not interactive, and are more efficient than generic secure multi-party computations.

Vaideep et al. [44] consider privacy friendly training of decision trees and forests. These techniques are orthogonal and complementary to our algorithms. We provide a zero-knowledge proof that a trained random forest classifier was applied on a secret data set correctly, and do not discuss the privacy of the training phase. Similarly, Magkos et al. [36] use protocols based on secure electronic election to implement a privacy friendly-frequency counter. They discuss how multiple rounds of their protocol can be used to privately train a random forest classifier.

Our cryptographic constructions build upon a number of previous cryptographic techniques, mostly involving zero-knowledge proofs, that we describe in detail in the next section.

4 Cryptographic Foundations

The novel techniques for resolving random forests and matching regular languages make use of modern cryptographic primitives, and in particular efficient zero-knowledge proofs of knowledge. We present here the basic building blocks, namely commitments, zero-knowledge proofs and signatures with efficient proof protocols (P-signatures). We also dwell deeper into how to use P-signatures to perform lookups into a public indexed table without revealing the lookup key or row retrieved.

4.1 Zero-knowledge, Commitments and P-signatures

Zero-Knowledge proofs. A zero-knowledge proof of knowledge [6] is a two-party protocol between a prover and a verifier. The prover demonstrates to the verifier her knowledge of some secret input (witness) that fulfills some statement without disclosing this input to the verifier. The protocol should fulfill two properties.

First, it should be a proof of knowledge, i.e., a prover without knowledge of the secret input convinces the verifier with negligible probability. Second, it should be zero-knowledge, i.e., the verifier learns nothing but the truth of the statement.

We make use of classical results for efficiently proving knowledge of discrete logarithm relations [42, 21, 19, 12, 10, 22]. To avoid common pitfalls, and to be compatible with the newest results in the literature, we use the language proposed by Camenisch et al. [15]. Their language is inspired by the CKY-language [14], which formalized and refined the PK notation for proofs of knowledge by Camenisch and Stadler [20]. While proofs of knowledge for the standard model are addressed in [14], Camenisch et al. [15] show how to realize proofs for their language in the UC-framework. As a third option, these proofs can be compiled [25] into non-interactive proofs in the random oracle model [7].

In the notation of [15], a protocol proving knowledge of integers $w_1, \ldots, w_n$ satisfying the predicate $\phi(w_1, \ldots, w_n)$ is described as

$$\text{ℵ}\, w_1 \in \mathcal{I}_1, \ldots, w_n \in \mathcal{I}_n : \phi(w_1, \ldots, w_n) \tag{1}$$

Here, we use the symbol "ℵ" instead of "∃" to indicate that we are proving "knowledge" of a witness, rather than just its existence.

The predicate $\phi(w_1, \ldots, w_n)$ is built up from "atoms" using arbitrary combinations of ANDs and ORs. An atom expresses *group relations*, such as $\prod_{j=1}^{k} g_j^{\mathcal{F}_j} = 1$, where the g_j are elements of an abelian group and the $\mathcal{F}_j$'s are integer polynomials in the variables $w_1, \ldots, w_n$.

Instead of using group relations directly, we rely on classical results that show how to reduce the following proof components to group relations: (1) linear relations and equality, (2) inequalities, and (3) proofs about commitments and signatures.

For the prime-order and the hidden-order setting there are different techniques for dealing with inequalities and P-signature possession. We refer to the literature for more details. We consciously keep our presentation independent of the lower-level cryptography by using an abstract notation for zero-knowledge statements.

Inequalities. A couple of techniques are available to prove that a value is positive. Groth [29], who builds on [9], relies on the fact that a number can be expressed using the sum of 3 squares. Alternatively, the value can be shown to be within the set of positive integers as provided by a trusted authority, using set membership techniques by Camenisch et al. [13].

Commitment schemes. A non-interactive commitment scheme consists of the algorithms $\mathsf{ComSetup}$, Commit and Open. $\mathsf{ComSetup}(1^k)$ generates the parameters of the commitment scheme par_c. $\mathsf{Commit}(par_c, x)$ outputs a commitment C_x to x and auxiliary information $open_x$. A commitment is opened by revealing $(x, open_x)$ and checking whether $\mathsf{Open}(par_c, C_x, x, open_x)$ outputs accept. The hiding property ensures that a commitment C_x to x does not reveal any information about x, whereas the binding property ensures that C_x cannot be

opened to another value x'. Our scheme requires commitment schemes that support an efficient proof predicates $\text{POpen}_{C_x}(x, open_x)$ for the opening algorithm of commitment C_x [37, 27].

P-signatures. A signature scheme consists of algorithms $(\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$. $\mathsf{Keygen}(1^k)$ outputs a key pair (sk, pk). $\mathsf{Sign}(sk, m)$ outputs a signature s on message m. $\mathsf{Verify}(pk, s, m)$ outputs accept if s is a valid signature on m and reject otherwise. This definition can be extended to support multi-block messages $\boldsymbol{m} = \{m_1, \ldots, m_n\}$. Existential unforgeability [28] requires that no probabilistic polynomial time (p.p.t.) adversary should be able to output a message-signature pair (s, m) unless he has previously obtained a signature on m.

One important proof component that we use in many of our constructions is a predicate $\text{PVerify}_{pk}(s, m_1, \ldots m_n)$ for the verification algorithm of a signature scheme. This allows to prove possession of a signature s, while keeping the signature itself and parts of the signed message secret. Signature schemes for which verification is efficiently provable are thus very attractive for protocol design, and are variously referred to as CL-signatures or P-signatures, i.e., signatures with efficient protocols [17, 18, 5].

Extended notation for zero-knowledge statements. We extend the notation above to abstract the details of proofs of knowledge that are used to verify private computations. In particular we allow the definition of named proof components that can be reused as sub-components of larger proofs.

We introduce the special notation $\text{F}(b) \rightarrow (a)$ to abstract details of proof components. We call this a proof component declaration. For example we may name a proof F on some secret inputs and outputs as being equivalent to a statement in zero-knowledge in the following manner: $\text{F}(a) \rightarrow (b) \equiv \nexists\, a, b : b = a + 1$. Semantically, the function F represents the proof that a counter was incremented. Secret a is the initial value and secret b the new value. Whether a variable appears on the left or the right is somewhat arbitrary and primarily meant to give useful intuition to users of the component. In terms of cryptography, it is simply syntactic sugar for the equivalent statement above.

Named proof components can be used in further higher-level proofs without their details being made explicit. For example, the proof $\nexists\, c, d : d = \text{F}(c)$ is equivalent to the two statements above. All variables within the component declaration (e.g. variables a, b in $\text{F}(a) \rightarrow (b)$) can be re-used in the high level proof. Any variables whose knowledge is proved, but that are not in the declaration, are considered inaccessible to the higher-level proof.

4.2 Direct Lookups

A P-signature on a sequence of messages, representing a number of keys and values, can be used to prove the equivalent of a table look-up [41].

Consider a public table T of keys k_i each mapping to the corresponding values v_i. Keys and values themselves can have multiple components $k_i = (k_{(i,0)}, \ldots, k_{(i,n-1)})$ and $v_i = (v_{(i,0)}, \ldots, v_{(i,m-1)})$.

A trusted authority A can encode this table to facilitate zero-knowledge lookups by providing a set of P-signatures t_i:

$$\forall i. \quad t_i = \mathsf{Sign}(\mathrm{sk}_A, \langle T_{id}, n, m, k_{(i,0)}, \ldots, k_{(i,n-1)}, v_{(i,0)}, \ldots, v_{(i,m-1)} \rangle) \tag{2}$$

To prove that a set of secrets (k', v') corresponds to a lookup in the table it is sufficient to show that:

$$\mathrm{LOOKUP}_T(k'_0, \ldots, k'_{n-1}) \rightarrow (v'_0, \ldots, v'_{m-1}) \equiv \tag{3}$$

$$\nexists\, t, k'_0, \ldots, k'_{n-1}, v'_0, \ldots, v'_{m-1} : \tag{4}$$

$$\mathrm{PVerify}_{pk_A}(t, T_{id}, n, m, k'_0, \ldots, k'_{n-1}, v'_0, \ldots, v'_{m-1}) \tag{5}$$

The predicate PVerify corresponds to the verification equations of the signature scheme. The proof hides the keys k' and corresponding values v', but not the identity of the table used to perform the lookup. To avoid any confusion, we always specify the table as part of the name of the lookup.

A variant of the proof allows for range lookups. In this case, a table T' contains as key a pair of values $(k_{(i,\min)}, k_{(i,\max)})$ defining a range, corresponding to a set of values $(v_{(i,0)}, \ldots, v_{(i,m-1)})$. We assume that the ranges defined by the keys do not overlap.

We denote the creation of the table with columns a, b and c, signed by authority A, by an algorithm called $\mathrm{ZKTABLE}_A([a_i, b_i, c_i])$. The cost of performing a lookup in terms of computation and communication is $O(1)$ and constructing a table requires $O(n)$ signatures in the size of the table.

5 Random Forests

Random forests are a state-of-the-art classification technique. A random forest is a collection of decision trees. Each decision tree is trained on a subset of a training dataset. Once the set of trees is trained, it can be used to classify unseen data items. Each decision tree is used separately to classify the item using its features and the majority decision is accepted as valid.

The aim of our protocol is to apply the random forest classification algorithm to a feature set that remains secret. At the same time, we wish to provide a proof that the classification task was performed correctly. We assume that the random forest classifier is already trained and public.

5.1 Vanilla Training and Resolution of Random Forests

This section provides a short overview of how a random forest is grown from a training set, as well as how it is used to classify data items, without any security considerations. Full details are available in Breiman [11]. The notation and resolution algorithms will be used as part of the privacy friendly protocols.

Consider a labelled training data set comprising items d_i. Each item has a set of M features, denoted as $F_m(d_i)$ for m in $[0, M-1]$. Each item is labelled with l_i into one of two classes c_0 or c_1.

The training algorithm takes two parameters: the depth D and the number of items N to be used to train each tree. Training each tree proceeds by sampling N items, with replacement, from the available data items d_i. Then, at each branch, starting with the root, a random feature is selected. The value of the feature that best splits the data items corresponding to this branch into the two separate classes is used to branch left or right. The algorithm proceeds to build further branches until the tree has reached the maximum depth D. Leaves store the decision or the relative number of items in each class.

A data item d that has not been previously seen is assigned a class by resolving each tree t in the forest. Starting at the root, the branching variable b and its threshold τ is used to decide whether to follow the right or left branch according to the corresponding feature $F_b(d')$. The process is repeated until a leaf is reached and the relative weight of belonging to two classes is stored as (c_{t0}, c_{t1}).

The sum of the weights corresponding to the two classes can be computed as $(\sum_t c_{t0}, \sum_t c_{t1})$. These sums represent the relative likelihood of the item belonging to the corresponding classes.

5.2 Encoding of Features and Trees

The key objective of our algorithm is to correctly execute the random forest classification algorithm on certified features without revealing the features. To enable this approach, both the user data and the random forest need to be cryptographically encoded in a specific manner.

First, the profile features are encoded as a table supporting zero knowledge lookups (as described in Section 4.2). A user u, with a profile with features k_i taking values v_i, has to be provided by an authority A with a table: $\mathrm{F}_u \equiv \mathrm{ZKTABLE}_A([k_i, v_i, u])$.

Second, we need to cryptographically encode each tree of the random forest. Each decision tree is individually encoded by building two tables supporting zero-knowledge lookups: one table for *branches* and one table for *leaves*.

Each non-leaf node of the decision tree is encoded as two separate left and right branches. Branches are encoded by an entry containing a node id n_i, a feature id k_i, and a range for the value $[v_{\min}, v_{\max}]$ as well as target node id n'_i. The node id n_i represents the node of the tree, while k_i is the feature used at this node to make a decision. If the branch feature is indeed in $[v_{\min,i}, v_{\max,i}]$, then the algorithm should proceed to n'_i. Representing an interval that has to be matched represents a uniform way to encode both left and right branches: left branches are encoded with an interval $[v_{\min}, v_{\max}] \equiv [\mathrm{MIN}_i, \tau_i - 1]$ and right branches with an interval $[v_{\min}, v_{\max}] \equiv [\tau_i, \mathrm{MAX}_i]$, where τ_i is the threshold of feature k_i that has a domain $[\mathrm{MIN}_i, \mathrm{MAX}_i]$. An authority A' encodes branches as: $\mathrm{B}_t \equiv \mathrm{ZKTABLE}_{A'}([n_i, k_i, v_{\min,i}, v_{\max,i}, n'_i])$.

Leafs are encoded separately to store the decision for each decision tree. Each leaf has a node id n_i and two values $c_{0,i}, c_{1,i}$ representing the relative likelihood of the item being in two classes respectively. The table representing leaves signed by A' is $\mathrm{L}_t \equiv \mathrm{ZKTABLE}_{A'}([n_i, c_{0,i}, c_{1,i}])$.

Branches and leaves of decision trees are encoded in order to certify its correctness when verifying proofs of signature possession. It is worth noting that they are public vis-a-vis the prover. In particular, the node id of the root node of each tree is known to all as is the fixed depth D of trees.

5.3 Private Resolution for Random Forests

Given a table of user features F_u, and a set of $|t|$ trees given by the sets of branch and leaf tables $\mathrm{B}_t, \mathrm{L}_t$, we present the proofs of correct classification.

In a nutshell, we can decompose the zero-knowledge proof required into performing the classification using the separate trees, and then aggregating and revealing the forest decision (without revealing the intermediate decisions). Each decision tree is matched by proving sequentially that the feature considered at each branch satisfies the prescribed condition.

The zero knowledge proof of correct resolution of a single tree is:

$$\begin{aligned}
&\mathrm{RESOLVETREE}_t(F_u, B_t, L_t) \rightarrow (c_{0,t}, c_{1,t}) \equiv && (6)\\
&\quad \nrightarrow \forall j. n_j, k_j, v_{\min,j}, v_{\max,j}, v_j, c_{0,t}, c_{1,t} : && (7)\\
&\quad n_0 = \mathrm{start} \wedge && (8)\\
&\quad \text{for } j = 0 \ldots D-2 : \{ && (9)\\
&\qquad n_j, k_j, v_{\min,j}, v_{\max,j}, n_{j+1} = \mathrm{LOOKUP}_{B_t}(n_j) \wedge && (10)\\
&\qquad v_j, u = \mathrm{LOOKUP}_{F_u}(k_j) \wedge && (11)\\
&\qquad v_{\min,j} \leq v_j \wedge v_j \leq v_{\max,j} \} \wedge && (12)\\
&\quad (c_{0,t}, c_{1,t}) = \mathrm{LOOKUP}_{L_t}(n_{D-1}) && (13)
\end{aligned}$$

The RESOLVETREE proof works by starting with a record of the known root node, and following $D-2$ branches to a leaf. For each branch, the proof demonstrates that it follows the previous branch, and that the feature variable concerned falls within the valid branch interval. The appropriate feature is looked-up in the feature table without revealing its identity or value. Note that n_0, u, D and j are not secret, which leads to some efficiency improvement when implementing the proof for the root branch.

To resolve a forest of $|t|$ trees, the following proof is required:

$$\begin{aligned}
&\mathrm{RESOLVEFOREST}_t(F_u, \forall t. B_t, L_t) \rightarrow (c_0, c_1) \equiv && (14)\\
&\quad \nrightarrow \forall t. c_{0,t}, c_{1,t}, c_0, c_1 : && (15)\\
&\quad \text{for } t = 0 \ldots |t|-1 : \{ && (16)\\
&\qquad (c_{0,t}, c_{1,t}) = \mathrm{RESOLVETREE}_t(F_u, B_t, L_t) \} \wedge && (17)\\
&\quad (c_0, c_1) = (\sum_t c_{0,t}, \sum_t c_{1,t}) \wedge && (18)\\
&\quad \mathrm{POpen}_{C_{c_0}}(c_0, \mathit{open}_{c_0}) \wedge \mathrm{POpen}_{C_{c_1}}(c_1, \mathit{open}_{c_1}) && (19)
\end{aligned}$$

The RESOLVEFOREST proof uses the tree resolution as a sub-component. The secret values returned from each tree are then aggregated into committed values

(c_0, c_1). These values can be revealed, or kept secret and used as part of a further protocol (for example to prove that the probability of the user belonging to a certain profile is higher than a set threshold).

5.4 Extensions

Decision forests can be used to classify items into more than two classes. To achieve this, the leaf table can encode likelihoods of multiple classes, and the aggregation algorithm can sum the evidence for more than two classes in parallel.

Similarly, more than one feature can be used to branch at each node. In such cases, each branch needs to encode the identity and interval for all necessary features. To maintain indistinguishability among all branches, the same number of features must be tested for all of them.

6 HMMs, Probabilistic Automata and Regular Languages

Regular languages can be used to match streams of symbols to a set of rules based on an initial state and possible transitions between states. Probabilistic automata extend regular languages to consider specific probabilities associated with initial states and transitions. Finally, hidden Markov models dissociate states from emitted symbols. We show how a client can match a sequence of certified actions or logs to a given model without revealing the specific actions or log entries. We illustrate our techniques using hidden Markov models (HMM), as both probabilistic automata and regular languages can be considered special cases of such models. We also discuss how matching simpler models can be implemented more efficiently.

6.1 Vanilla Matching for Hidden Markov Models

We first review the HMM matching algorithms in the absence of any privacy protection. A complete tutorial on hidden Markov model based techniques can be found in [39].

A hidden Markov model is fully defined by three tables: an *initial state* table, a *transition* table, and a *symbol emission* table. All tables refer to a finite set of hidden states and observed symbols. The initial state table maps possible initial states of the model to a probability. The state transition table maps state pairs to a probability. Finally, the symbol emission table maps state and symbol tuples to a probability. We assume that any entry missing from a table is implicitly assigned a probability of zero. By convention, a specific state is considered to be the final state of a model, and matching ends there.

Matching a sequence of symbols starts with assigning the first symbol both a state and a probability from the initial state table, and, given the initial state, a probability of emitting the observed first symbol from the symbol emission table. Matching then proceeds through the sequence of observed symbols by

selecting a new hidden state for each symbol. The probability of state transition and the probability of emitting the subsequent symbol from the new state can be retrieved from the state transition table and the symbol emission table respectively.

The total probability of a match is the product of all the probabilities from the initial state to the final state. This product can also be expressed as the sum of the logarithms of all the probabilities.

6.2 Encoding of Hidden Markov Models

To enable a user to prove that a sequence of their actions or log is matched by a hidden Markov model, both their actions and the HMM need to be encoded in a specific manner.

First, we consider how an authority A encodes a sequence of symbols relating to a user. This sequence can represent specific user actions or generic log entries relating to a user u. Each symbol in the sequence is represented by an individual entry within a table supporting zero-knowledge lookups $\mathrm{S}_u \equiv \mathrm{ZKTABLE}_A([t_i, e_i, u])$, where t_i is a sequence number and e_i represents the symbol observed.

Each table of the hidden Markov model is also encoded using tables that allow for zero-knowledge lookups signed by authority A'. The initial symbol table I_t contains: $\mathrm{I}_t \equiv \mathrm{ZKTABLE}_{A'}([s_i, p_i])$, where s_i is a possible initial state and p_i the logarithm of its probability. The state transition table is represented as: $\mathrm{T}_t \equiv \mathrm{ZKTABLE}_{A'}([s_i, s_j, p_i])$, where s_i and s_j are states between which there is a transition with log-probability p_i. Finally, the emission table for each state s_i and potential emitted symbol e_j is encoded as: $\mathrm{E}_t \equiv \mathrm{ZKTABLE}_{A'}([s_i, e_j, p_i])$.

We implicitly assign a zero probability to any entries that are not encoded in the above tables. No proof of a match can be produced for zero-probability matches. Furthermore, we store all probabilities as a positive integer representing the quantised version of their negative logarithm. Therefore, the expression p_i used above is a shorthand for $-\lceil 10^{\psi} \log \pi_i \rceil$ in case ψ decimal digits of the logarithm are to be used where π_i is the raw probability.

6.3 Private Matching

A user with a certified sequence of actions S_u can prove that it matches a hidden Markov model described by a triplet of tables (I_t, T_t, E_t). This can be done without revealing the symbols in the sequence of actions, their position within the sequence and without revealing the matched sequence or states or matching probability. In practice, it might be necessary to reveal the position of the match, the probability of the match or some hidden states for subsequent processing. We present the algorithms where only commitments to these are produced, which can be selectively opened to reveal interesting values.

The MATCH function proves that a sequence of hidden states $s_0 \dots s_{l-1}$ match the sequence of symbols $e_0 \dots e_{l-1}$ present, at the offset t_0 of the user

sequence, with probability p.

$$\text{MATCH}(S_u, I_t, T_t, E_t) \rightarrow (s_0 \dots s_{l-1}, e_0 \dots e_{l-1}, t_0, p) \equiv \tag{20}$$
$$\nexists \forall i \in [0, l-1]. t_i, s_i, e_i, p_i, p'_i, p : \tag{21}$$
$$e_0, u = \text{LOOKUP}_{S_u}(t_0) \wedge \tag{22}$$
$$p_0 = \text{LOOKUP}_{I_t}(s_0) \wedge \tag{23}$$
$$p'_0 = \text{LOOKUP}_{E_t}(s_0, e_0) \wedge \tag{24}$$
$$\text{for } i = 1 \dots |i| - 1 : \{ \tag{25}$$
$$t_i = t_0 + i \wedge \tag{26}$$
$$e_i, u = \text{LOOKUP}_{S_u}(t_i) \wedge \tag{27}$$
$$p_i = \text{LOOKUP}_{T_t}(s_{i-1}, s_i) \wedge \tag{28}$$
$$p'_i = \text{LOOKUP}_{E_t}(s_i, e_i)\} \wedge \tag{29}$$
$$p = \sum_{0}^{l-1} (p_i + p'_i) \wedge \tag{30}$$
$$\text{POpen}_{C_p}(p, open_p) \wedge \text{POpen}_{C_{s_{l-1}}}(s_{l-1}, open_{s_{l-1}}) \tag{31}$$

The main result of the protocol is a commitment $C_{s_{l-1}}$ to the final state s_{l-1} and a commitment C_p to the probability p of reaching this state. The prover can open $C_{s_{l-1}}$ and C_p. The raw probability of a match can be computed as $\pi = e^{10^{-\psi} \cdot -p}$. Instead of opening the commitment corresponding to the probability of the match p the user could prove in zero knowledge that it is within a certain range.

We note that we do not hide the length of the match denoted by l. The user is also free to commit to and open commitments to any of the values $s_0 \dots s_{l-1}$, $e_0 \dots e_{l-1}$, t_0 to reveal more information to the verifier.

We note that the above procedure proves there is a match and returns its probability, but does not guarantee in itself that this is the best match, i.e. the match with the highest probability. Such a procedure is likely to be computationally more expensive, and thus not really practical for real-world applications. In comparison, the above match algorithm requires only a linear number of proofs in the length of the match.

6.4 Simplifications for Finite Automata and Regular Languages

Using the match algorithm for HMMs, we can trivially match regular languages and probabilistic automata. For regular languages, we assume that each state only emits one symbol with probability one, and that all probability values are equal. For finite automata, we extend the encoding for regular languages to allow for different probabilities for the initial state and transitions between states. In both cases, the matching algorithm can be simplified to avoid proving the link between symbols and states, as they are in fact the same entities.

Similarly, for regular languages there is no need to store or process probabilities as the length of the match contains all the necessary information.

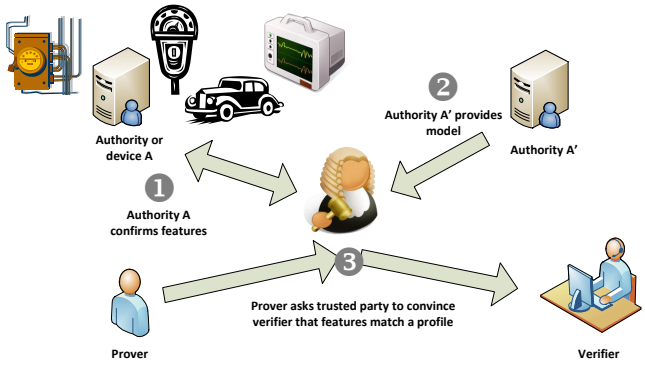

Fig. 2. An ideal protocol with a trusted party.

7 Discussion

7.1 Security

Security and privacy properties. The security and privacy properties of our client-side profiling schemes can be described via an ideal protocol depicted in Figure 2 that carries out the same task. In the ideal protocol, a trusted party T receives all the inputs of the parties involved in the protocol, performs the required computations, and provides each party with its output.

As mentioned in Section 2, we have a setting with four parties: provers P, verifiers V, an authority A that certifies prover's profiles, and an authority A' that certifies the model. In the ideal protocol, first, A' sends T the model, which is either the random forest or the hidden Markov model. P sends T its features, which forwards them to A. A replies T whether the features are correct, and, if it is the case, T stores them. When P tells T to send the result of the profiling task to V, T employs P's features and the received model to perform the profiling task and sends V the result.

As can be seen, in the ideal protocol V only learns the result, which protects P's privacy. The resolution of the profiling task is performed by T, which is trusted, and thus the correctness of the result is guaranteed.

Our scheme mirrors the security and privacy properties of the ideal protocol while limiting the involvement of trusted parties. Namely, our protocols require A and A' to be trusted when certifying the correctness of provers' profiles and of the model respectively. Once this is done, A and A' do not participate in the protocol anymore. P and V can be adversarial, and yet our scheme guarantees that, as in the ideal protocol, the result computed by P is correct and V does not learn more information on P's features. We note that our protocols reveal to V a small amount of information on the model, such as the depth of the decision

trees. Such information is also revealed to V by the ideal functionality in the ideal protocol.

Secure implementation of our protocols. It may seem that the provable guarantees afforded by zero-knowledge proofs of knowledge, particularly when using the compilation techniques of [15] to generate a universally composable protocol, are all that is needed to implement such an ideal functionality, and thus to automatically guarantee the authenticity and privacy of the classification result. The soundness of proofs guarantees the correctness and authenticity of the classification task towards verifiers, and the fact that the protocols are zero-knowledge guarantees to users that nothing besides the result of the classification is revealed.

Our system is, however, not only a zero-knowledge proof of knowledge. It also consists of input generators that provide auxiliary information for the proofs, e.g., signed lookup tables. Unfortunately, as noted by [14] many zero-knowledge proof protocols designed with only honestly generated input in mind, are not portable, i.e., they do not necessarily retain their properties when executed on malicious input. In our system the input generation stage involves the prover P, verifiers V, an authority A that certifies users profiles, and an authority A' that certifies the tables and algorithms used for classification. When considering *authenticity* A and A' are considered as being fully trusted by the verifier. For authenticity, portability is thus naturally maintained. For *privacy* we consider A and A', however, as potentially colluding with the verifier. To prevent the attacks of [33] the prover needs to be able to check that the cryptographic group parameters provided by A and A' are well formed as required in [14]. In our case these parameters primarily consist of P-signature public keys, and commitment parameters. To meet their security properties these schemes should already be carefully designed to be portable. Note that provers need to verify the correct generation of these public keys and commitment parameters only once in the *setup* phase.

Portability is significantly easier for prime order groups. Also in light of [34], using a single pairing-based elliptic curve could thus be a conscious design decision, if one is willing to live without Strong RSA based CL-signatures (there are pairing-based alternatives [18]) and sum-of-squares based range proofs [9]. This may, however, exclude credential issuing authorities [16] that only support the more mature Strong RSA based CL-signatures.

We do not provide a concrete implementation and leave many low-level design decisions underspecified. We advice against adhoc implementations, and recommend the use of zero-knowledge compilation frameworks such as [3, 1] that are approaching maturity.

Systems security issues. A malicious service provider could utilize our client-side profiling schemes and yet employ covertly web-bugs, cookies or other techniques to obtain further information about users. We point out that existing countermeasures, e.g., [2] for web-bugs, can be employed to prevent that. When used in

combination with out scheme, service providers still get the result of the profiling task.

7.2 Efficiency of Proposed Schemes

Encoding a table for zero-knowledge lookups using $\text{ZKTABLE}_A(\cdot)$ requires one re-ranomizable signature per encoded row. The LOOKUP operation requires the proof of possession of a single P-signature, and RLOOKUP additionally requires a range proof to ensure the key is within a secret range.

To prove that a value x lies within an interval $[a, b]$, it is necessary to show that $x - a \geq 0$ and $b - x \geq 0$. The techniques of Groth [29] requiring 3 additional commitments and 3 zero-knowledge proofs of multiplication. Using set membership techniques by Camenisch et al. [13] requires two proof of possession of a signature per range proof.

The RESOLVETREE procedure relies heavily on zero-knowledge lookups and range proofs: two LOOKUP and a range proof are needed per branch followed in the tree. A single lookup is needed to retrieve the leaf with the decision. In total $4 \cdot (D - 1) + 1$ proofs of possession of a signature are needed to resolve each decision tree, where D is the depth of the tree. Aggregating the decisions of multiple trees in RESOLVEFOREST can be done very efficiently using homomorphic properties of commitments.

The MATCH procedure to resolve hidden Markov models requires 3 lookups per symbol matched in the general case. This can be reduced to two lookups if only regular automata are used (and the emitted symbols are equivalent to the matched states). In case the location of the sequence into the user stream is not secret, the stream can be encoded as a sequence of commitments, and a single lookup is necessary per matched symbol.

8 Conclusion

A rule for building secure system states that all user input must be positively validated to be correct. Traditionally, any computation performed on the client-side cannot be trusted. As a result, personal information must be available to be processed on the server side, which leads to a number of privacy problems.

In this paper, we extend the paradigm of secure client-side computation [23], and show that complex classification and pattern recognition tasks on user secrets can be proved correct. We heavily rely on the efficiency of proving possession of P-signatures to efficiently implement random forest and hidden Markov model matching.

Our techniques allow for classification and matching without any unnecessary leakage of other information. Matching and classification are often necessary to transform unstructured data into information that can be acted upon: to perform authentication, to tune a business decision, to detect intrusion, to reason about a stream of activity.

Yet, in many contexts *the mere act of profiling* in itself can violate privacy. Recent news stories, for example, describe how data mining techniques can be used to detect whether one is pregnant for the purposes of marketing from shifts in their buying habits[1]. Our techniques could deliver comparative functionality without the need to reveal the detailed shopping list of a customer. The extent to which one should be subjected to such profiling at all is highly debatable.

Providing a definite answer as to the necessity or oppression inherent in profiling is beyond the scope of this work. Those applying it to specific settings would be advised to reflect on the remarks Roland Barthes made at his inaugural lecture at the *College de France*: "*We do not see the power which is in speech because we forget that all speech is a classification, and that all classifications are oppressive.*"[2]

Acknowledgements. The authors would like to thanks the anonymous reviewers and the paper shepherd, Ayday Erman, for suggestions that improved this work.

References

1. J. B. Almeida, E. Bangerter, M. Barbosa, S. Krenn, A.-R. Sadeghi, and T. Schneider. A certifying compiler for zero-knowledge proofs of knowledge based on sigma-protocols. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 151–167. Springer, 2010.
2. A. Alsaid and D. Martin. Detecting web bugs with bugnosis: Privacy advocacy through education. In R. Dingledine and P. F. Syverson, editors, *Privacy Enhancing Technologies*, volume 2482 of *Lecture Notes in Computer Science*, pages 13–26. Springer, 2002.
3. E. Bangerter, T. Briner, W. Henecka, S. Krenn, A.-R. Sadeghi, and T. Schneider. Automatic generation of sigma-protocols. In F. Martinelli and B. Preneel, editors, *EuroPKI*, volume 6391 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2009.
4. L. E. Baum and T. Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.
5. M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *TCC*, pages 356–374, 2008.
6. M. Bellare and O. Goldreich. On defining proofs of knowledge. In E. F. Brickell, editor, *CRYPTO '92*, volume 740, pages 390–420. Springer-Verlag, 1992.
7. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communication Security*, pages 62–73. Association for Computing Machinery, 1993.
8. A. Bosch, A. Zisserman, and X. Muoz. Image classification using random forests and ferns. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. Ieee, 2007.

[1] Charles Duhigg. How Companies Learn Your Secrets. The New York Times. February 16, 2012.

[2] In Barthes: Selected Writings (1982). Lecon (1978).

9. F. Boudot. Efficient proofs that a committed number lies in an interval. LNCS, pages 431–444, 2000.
10. S. Brands. Rapid demonstration of linear relations connected by boolean operators. In W. Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *LNCS*, pages 318–333. Springer Verlag, 1997.
11. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
12. J. Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zürich, 1998.
13. J. Camenisch, R. Chaabouni, and A. Shelat. Efficient protocols for set membership and range proofs. In J. Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 234–252. Springer, 2008.
14. J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. In *EUROCRYPT*, pages 425–442, 2009.
15. J. Camenisch, S. Krenn, and V. Shoup. A framework for practical universally composable zero-knowledge protocols. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 449–467. Springer, 2011.
16. J. Camenisch and A. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In B. Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer Verlag, 2001.
17. J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *SCN*, pages 268–289, 2002.
18. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. K. Franklin, editor, *Advances in Cryptology — CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer Verlag, 2004.
19. J. Camenisch and M. Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In J. Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *LNCS*, pages 107–122. Springer Verlag, 1999.
20. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. Kaliski, editor, *CRYPTO '97*, volume 1296 of *LNCS*, pages 410–424. Springer Verlag, 1997.
21. D. Chaum and T. Pedersen. Wallet databases with observers. In *CRYPTO '92*, volume 740 of *LNCS*, pages 89–105, 1993.
22. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.
23. G. Danezis and B. Livshits. Towards ensuring client-side computational integrity. In C. Cachin and T. Ristenpart, editors, *CCSW*, pages 125–130. ACM, 2011.
24. R. Díaz-Uriarte and S. De Andres. Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):3, 2006.
25. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. Odlyzko, editor, *CRYPTO*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.
26. M. Fredrikson and B. Livshits. Repriv: Re-imagining content personalization and in-browser privacy. In *IEEE Symposium on Security and Privacy*, pages 131–146. IEEE Computer Society, 2011.
27. E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. LNCS, pages 16–30, 1997.
28. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

29. J. Groth. Non-interactive zero-knowledge arguments for voting. In J. Ioannidis, A. Keromytis, and M. Yung, editors, *ACNS*, volume 3531 of *Lecture Notes in Computer Science*, pages 467–482, 2005.
30. S. Guha, B. Cheng, and P. Francis. Privad: Practical Privacy in Online Advertising. In *Proceedings of the 8th Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, Mar 2011.
31. B. Juang. Hidden markov models. *Encyclopedia of Telecommunications*, 1985.
32. K. Karplus, C. Barrett, and R. Hughey. Hidden markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–856, 1998.
33. S. Kunz-Jacques, G. Martinet, G. Poupard, and J. Stern. Cryptanalysis of an efficient proof of knowledge of discrete logarithm. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 27–43. Springer, 2006.
34. A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, whit is right. Cryptology ePrint Archive, Report 2012/064, 2012. `http://eprint.iacr.org/`.
35. B. N. Levine, C. Shields, and N. B. Margolin. A survey of solutions to the sybil attack, 2006.
36. E. Magkos, M. Maragoudakis, V. Chrissikopoulos, and S. Gritzalis. Accurate and large-scale privacy-preserving data mining using the election paradigm. *Data & Knowledge Engineering*, 68(11):1224–1236, 2009.
37. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. LNCS, pages 129–140, 1992.
38. B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *SIGKDD Explorations*, 4(2):12–19, 2002.
39. L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
40. A. Reznichenko, S. Guha, and P. Francis. Auctions in Do-Not-Track Compliant Internet Advertising. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, Chicago, IL, Oct 2011.
41. A. Rial and G. Danezis. Privacy-Preserving Smart Metering. In *Proceedings of the 11th ACM workshop on Privacy in the electronic society (WPES 2011)*, page 12, Chicago,IL,USA, 2011. ACM.
42. C. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.
43. V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *NDSS*, 2010.
44. J. Vaidya, C. Clifton, M. Kantarcioglu, and A. S. Patterson. Privacy-preserving decision trees over vertically partitioned data. *TKDD*, 2(3), 2008.