Privacy-Oriented Data Mining by Proof Checking

Amy Felty[1] and Stan Matwin[2]*

[1] SITE, University of Ottawa, Ottawa, Ontario K1N 6N5, Canada
`afelty@site.uottawa.ca`
[2] LRI – Bt 490, Université Paris-Sud, 91405 ORSAY CEDEX, France
`stan@site.uottawa.ca`

Abstract. This paper shows a new method which promotes ownership of data by people about whom the data was collected. The data owner may preclude the data from being used for some purposes, and allow it to be used for other purposes. We show an approach, based on checking the proofs of program properties, which implements this idea and provides a tool for a verifiable implementation of the Use Limitation Principle. The paper discusses in detail a scheme which implements data privacy following the proposed approach, presents the technical components of the solution, and shows a detailed example. We also discuss a mechanism by which the proposed method could be introduced in industrial practice.

1 Introduction

Privacy protection, generally understood as "...the right of individuals to control the collection, use and dissemination of their personal information that is held by others" [5], is one of the main issues causing criticism and concern surrounding KDD and data mining. Cheap, ubiquitous and persistent database and KDD resources, techniques and tools provide companies, governments and individuals with means to collect, extract and analyze information about groups of people and individual persons. As such, these tools remove the use of person's data from their control: a consumer has very little control over the uses of data about her shopping behavior, and even less control over operations that combine this data with data about her driving or banking habits, and perform KDD-type inferences on those combined datasets. In that sense, there is no data ownership by the person whom the data is about. This situation has been the topic of growing concern among people sensitive to societal effects of IT in general, and KDD in particular. Consequently, both macro-level and technical solutions have been proposed by the legal and IT community, respectively.

At the macro-level, one of the main concepts is based on the fact that in most of the existing settings the data collectors are free to collect and use data as long as these operations are not violating constraints explicitly stated by the individuals whose data are used. The onus of explicitly limiting the access to

* On leave from SITE, University of Ottawa, Canada

one's data is on the data owner: this approach is called "opting-out". It is widely felt (e.g. [8]) that a better approach would be opting-in, where data could only be collected with an explicit consent for the collection and specific usage from the data owner.

Another macro concept is the Use Limitation Principle (ULP), stating that the data should be used only for the explicit purpose for which it has been collected. It has been noted, however, that "...[ULP] is perhaps the most difficult to address in the context of data mining or, indeed, a host of other applications that benefit from the subsequent use of data in ways never contemplated or anticipated at the time of the initial collection." [7].

At the technical level, there has been several attempts to address privacy concerns related to data collection by websites, and subsequent mining of this data. The main such proposal is the Platform for Privacy Preferences (P3P) standard, developed by the WWW Consortium [11]. The main idea behind the P3P is a standard by which websites, collecting data from the users, will describe their policies and ULPs in XML. Users, or their browsers, will then decide whether the site's data usage is consistent with the user's constraints on the use of their data, which can also be specified as part of a P3P privacy specification. Although P3P seems to be a step in the right direction, it has some well-known shortcomings. Firstly, the core of a P3P definition for a given website is the description of a ULP, called the P3P policy file. This policy file describes, in unrestricted natural language, what data is collected on this site, how it is used, with whom it is shared, and so on. There are no provisions for enforcement of the P3P policies, and it seems that such provisions could not be incorporated into P3P: the policy description in natural language cannot be automatically verified. The second weakness, noted by [6], is the fact that while P3P provides tools for opting-out, it does not provide tools for opting-in.

The data mining community has devoted relatively little effort to address the privacy concerns at the technical level. A notable exception is the work of R. Agrawal and R. Srikant [2]. In that paper the authors propose a procedure in which some or all numerical attributes are perturbed by a randomized value distortion, so that both the original values and their distributions are changed. The proposed procedure then performs a reconstruction of the original distribution. This reconstruction does not reveal the original values of the data, and yet allows the learning of decision trees which are comparable in their performance to the trees built on the original, "open" data. A subsequent paper [1] shows a reconstruction method which, for large data sets, does not entail information loss with respect to the original distribution. Although this proposal, currently limited to real-valued attributes (so not covering personal data such as SSN, phone numbers etc.) goes a long way towards protecting private data of an individual, the onus of perturbing the data and guaranteeing that the original data is not used rests with the organization performing the data mining. There is no mechanism ensuring that privacy is indeed followed by them.

In this paper we propose a different approach to enforce data ownership, understood as full control of the use of the data by the person whom the data describes. The proposed mechanism can support both the opt-out and the opt-in approach to data collection. It uses a symbolic representation of policies, which makes policies enforceable. Consequently, the proposed approach is a step into the direction of verifiable ULPs.

2 Overall Idea

The main idea of the proposed approach is the following process:

1. Individuals make symbolic statements describing what can and/or cannot be done with specific data about them. These permissions are attached to their data.
2. Data mining and database software checks and respects these statements.

In order to obtain a guarantee that 2) holds regardless of who performs the data mining/database operations, we propose the following additional steps:

a. Data mining software developers provide, with their software, tools and building blocks with which users of the software can build theorems in a formal language (with proofs), stating that the software respects the user's permissions.
b. An independent organization is, on request, allowed (remote) access to the environment that includes the data mining software and the theorems with the proofs, and by running a proof checker in this environment it can verify that the permissions are indeed respected by the software.

We can express this idea more formally as a high-level pseudo-code, to which we will refer in the remainder of the paper. We assume that in our privacy-oriented data mining scenario there are the following players and objects:

1. C, an individual (viewed at the same time as a driver, a patient, a student, a consumer, etc.)
2. a set of databases D containing records on different aspects of C's life, e.g. driving, health, education, shopping, video rentals , etc.
3. a set of algorithms and data mining procedures A, involving records and databases from D, e.g. join two database tables, induce a classification tree from examples, etc.
4. a set (language) of permissions P for using data. P is a set of rules (statements) about elements of D and A. C develops her own set of permissions by choosing and/or combining elements of P and obtains a P_C. P_C enforces C's ownership of the data. e.g. "my banking record shall not be cross-referenced (joined) with my video rental record" or "I agree to be in a decision tree leaf only with at least 100 other records". Here, we view P_C as a statement which is (or can be translated to) a predicate on programs expressed in a formal logic.
5. Org is an organization, e.g. a data mining consultancy, performing operations $a \in A$ on a large dataset containing data about many Cs.

6. S is the source code of $a \in A$, belonging to the data mining tool developer, and B is the executable of S. S may reside somewhere else than at Org's, while B resides with Org.
7. A certifiable link $L(B, S)$ exists between B and S, i.e. Org and $Veri$ (see below) may verify that indeed $S =$ source code of B.
8. $P_C(S)$ is then a theorem stating that S is a program satisfying constraints and/or permissions on the use of data about C.
9. H is a proof checker capable of checking proofs of theorems $P_C(S)$.
10. $Veri$, a Verifier, is a generally trusted organization whose mandate here is to check that Org does not breach C's permission.

The following behavior of the players C, Org and $Veri$ in a typical data mining exercise is then provably respectful of the permissions of C with respect to their data:

1. Org wants to mine some data from D (such that C's records are involved) with B. This data is referred to as $data_C$.
2. $data_C$ comes packaged with a set of C's permissions: $data_C \mid\mid P_C$.
3. Org was given by the data mining tool developer (or Org itself has built) a proof $R(S, P_C)$ that S respects P_C whenever C's data is processed. Consequently, due to 7) above, B also respects P_C.
4. Org makes $R(S, P_C)$ visible to $Veri$.
5. $Veri$ uses P_C and S to obtain $P_C(S)$, and then $Veri$ uses H to check that $R(S, P_C)$ is the proof of $P_C(S)$, which means that S guarantees P_C.

We can observe that, by following this scheme, $Veri$ can verify that any permissions stated by C are respected by Org (more exactly, are respected by the executable software run by Org). Consequently, we have a method in which any ULP, expressed in terms of a P_C, becomes enforceable by $Veri$. The P_C permissions can be both "negative", implementing an opt-out approach, and "positive", implementing an opt-in approach. The latter could be done for some consideration (e.g. a micropayment) of C by Org.

It is important to note that in the proposed scheme there is no need to consider the owner of the data D: in fact, D on its own is useless because it can only be used in the context of P_Cs of the different Cs who are described by D. We can say that Cs represented in D effectively own the data about themselves. Another important comment emphasizes the fact that there are two theorem proving activities involved in the proposed approach: proof construction is done by Org or the data mining developer, and proof checking is done by $Veri$. Both have been the topics of active research for more than four decades, and for both, automatic procedures exist. In our approach, we rely on an off-the-shelf solution [10] described in the next section. Between the two, the relatively hard proof construction is left as a one-time exercise for Org or for the tool developer where it can be assisted by human personnel, while much easier and faster automatic proof checking procedure is performed by the proposed system.

Let us observe that $Veri$ needs the $data_C \mid\mid P_C$, $R(S, P_C)$, and S. S will need to be obtained from the data mining tool developer, and P_C can be obtained from C. Only $R(S, P_C)$ is needed from Org (access to B will also be needed for the purpose of checking $L(B, S)$). In general, $Veri$'s access to Org needs to be minimal for the scheme to be acceptable to Org. In that respect, we can observe that $Veri$ runs proof checking H on a control basis, i.e. not with every execution of B by Org, but only occasionally, perhaps at random time intervals, and even then only using a randomly sampled C. A brief comment seems in order to discuss the performance of the system with a large number of users, i.e. when A works on a D which contains data about many Cs. The overhead associated with the processing many Cs is linear in their number. In fact, for each $C \in D$ this overhead can be conceptually split into two parts: 1. the proof checking part (i.e. checking the proof of $P_C(S)$), and 2. the execution part (i.e. extra checks resulting from C's permissions are executed in the code B). The first overhead, which is the expensive one, needs to be performed only once for each C involved in the database. This could be handled in a preprocessing run.

At the implementation level, H could behave like an applet that Org downloads from $Veri$.

3 Implementation

Our current prototype implementation uses the Coq Proof Assistant [10]. Coq implements the Calculus of Inductive Constructions (CIC), which is a highly expressive logic. It contains within it a functional programming language that we use to express the source code, S, of the data mining program examples discussed in this paper. Permissions, P_C, are expressed as logical properties about these programs, also using CIC. The Coq system is interactive and provides step-by-step assistance in building proofs. We discuss its use below in building and checking a proof $R(S, P_C)$ of a property P_C of an example program S.

Our main criterion in choosing a proof assistant was that it had to implement a logic that was expressive enough to include a programming language in which we could write our data mining programs, and also include enough reasoning power to reason about such programs. In addition, the programming language should be similar to well-known commonly used programming languages. Among the several that met these criteria, we chose the one we were most familiar with.

3.1 Proof Checking

Proof checking is our enforcement mechanism, insuring that programs meet the permissions specified by individuals. In Coq, proofs are built by entering commands, one at a time, that each contribute to the construction of a CIC term representing the proof. The proof term is built in the background as the proving process proceeds. Once a proof is completed, the term representing the complete proof can be displayed and stored. Coq provides commands to replay and compile completed proofs. Both of these commands include a complete check of the proof. $Veri$ can use them to check whether a given proof term is indeed a proof of a specified property. Coq is a large system, and the code for checking proofs is only a small part of it. All of the code used for building proofs need not be trusted since the proof it builds can be checked after completion. The code for checking proofs is the only part of our enforcement mechanism that needs to be trusted. As stated, $Veri$ is a generally trusted organization, so to ensure this trust $Veri$ must certify to all others that it trusts the proof checker it is using, perhaps by implementing it itself.

3.2 Verifiable Link Between the Source Code and the Object Code

As pointed out in Sect. 2, the scheme proposed here relies on a verifiable link $L(B, S)$ between the source code and the object code of the data mining program. Since theorems and proofs refer to the source programs, while the operations are performed by the object program, and the source S and object B reside with different players of the proposed scheme, we must have a guarantee that all the properties obtained for the source program are true for the code that is actually executed. This is not a data mining problem, but a literature search and personal queries did not reveal an existing solution, so we propose one here.

In a simplistic way, since $Veri$ has access to S, $Veri$ could compile S with the same compiler that was used to obtain B and compare the result with what Org is running. But compilation would be an extremely costly operation to be performed with each verification of $L(B, S)$. We propose a more efficient scheme, based on the concept of digital watermarking. S, which, in practice, is a rich library structure, containing libraries, makefiles etc., is first `tar`'ed. Then the resulting sequential file $\texttt{tar}(S)$ is hashed by means of one of the standard hash functions used in the Secure Sockets Layer standard SSL, implemented in all the current Internet browsers. The Message Digest function MD5 [9] is an example of such a file fingerprinting function. The resulting, 128-bit long fingerprint of S is then embedded in random locations within B in the form of `DO NOTHING` instructions whose address part is filled with the consecutive bits forming the result of MD5.

This encoding inside B will originally be produced by a compiler, engineered for this purpose. Locations containing the fingerprint—a short sequence of integer numbers—are part of the definition of $L(B, S)$ and are known to $Veri$. $Veri$ needs to produce $\mathrm{MD5}(\texttt{tar}(S))$ and check these locations within B accordingly. The whole process of checking of $L(B, S)$ can be performed by a specialized applet, ensuring that B is not modified or copied.

3.3 Permissions Language

The logic implemented by Coq is currently used as our language of permissions. More specifically, any predicate expressible in Coq which takes a program as an argument is currently allowed. Each such predicate comes with type restrictions on the program. It specifies what the types of the input arguments to the program must be, as well as the type of the result. An example is given in the next section.

3.4 Issues

The permissions language is currently very general. We plan to design a language that is easy for users to understand and use, and can be translated to statements of theorems in Coq (or some other theorem prover).

As mentioned, a proof in Coq is built interactively with the user supplying every step. Having a smaller permissions language targeted to the data mining application will allow us to clearly identify the class of theorems we want to be able to prove. We will examine this restricted class and develop techniques for automating proof search for it, thus relieving much of the burden of finding proofs currently placed on either the data mining tool developer or Org. These automated search procedures would become part of the tools and building blocks provided by data mining software developers.

In our Coq solution described so far, and illustrated by example in the next section, we implement source code S using the programming language in Coq. We actually began with a Java program, and translated it by hand to Coq so that we could carry out the proof. In practice, proofs done directly on actual code supplied by data mining software developers would be much more difficult, but it is important to keep a connection between the two. We would like to more precisely define our translation algorithm from Java to Coq, and automate as much of it as possible. For now, we propose that the data mining tool developers perform the translation manually, and include a description of it as part of the documentation provided with their tools.

In the domain of Java and security, Coq has also been used to reason about the JavaCard programming language for multiple application smartcards [3], and to prove correctness properties of a Java byte-code verifier [4].

4 Example

We present an example program which performs a database join operation. This program accommodates users who have requested that their data not be used in a join operation by ignoring the data for all such users; none of their data will be present in the data output by the program. We present the program and discuss the proof in Coq. We first present the syntax of the terms of CIC used here. Let x and y represent variables and M, N represent terms of CIC. The class of CIC terms are defined using the following grammar.

$$\begin{gathered} Prop \ \mid \ Set \ \mid \\ M = N \ \mid \ M \wedge N \ \mid \ M \vee N \ \mid \ M \rightarrow N \ \mid \ \neg M \ \mid \ \forall x : M.N \ \mid \ \exists x : M.N \\ x \ \mid \ MN \ \mid \ [x : M]N \ \mid \ x\ \{y_1 : M_1; \ldots ; y_n : M_n\} \ \mid \\ Case\ x : M\ of\ M_1 \Rightarrow N_1, \ldots , M_n \Rightarrow N_n \end{gathered}$$

This set of terms includes both logical formulas and terms of the functional programming language. $Prop$ is the type of logical propositions, whereas Set is the type of data types. For instance, two data types that we use in our example are the primitive type for natural numbers and user-defined records. In Coq these types are considered to be members of Set. All the usual logical connectives for well-formed formulas are found on the second line. Note that in the quantified formulas, the type of the bound variable, namely M is given explicitly. N is the rest of the formula which may contain occurrences of the bound variable. CIC is a higher-order logic, which means for instance, that quantification over predicates and functions is allowed. On the third line, MN represents application, for example of a function or predicate M to its argument N. We write $MN_1 \ldots N_n$ to represent $(((MN_1)\ldots)N_n)$. The syntax $[x : M]N$ represents a parameterized term. For instance, in our example, N often represents a function that takes an argument x of type M.

The term $x\ \{y_1 : M_1; \ldots; y_n : M_n\}$ allows us to define record types, where $y_1, \ldots, y_n$ are the field names, $M_1, \ldots, M_n$ are their types, and x is the name of the constant used to build records. For example, a new record is formed by writing $xN_1, \ldots, N_n$, where for $i = 1, \ldots, n$, the term N_i has type M_i and is the value for field y_i. For our example program, we will use three records. One of these records, for example is the following used to store payroll information.

$$\begin{array}{l} \textsf{Record}\ \mathit{Payroll} : \mathit{Set} := \\ \quad \mathit{mkPay}\ \{\mathit{PID} : \mathit{nat};\ \mathit{JoinInd} : \mathit{bool};\ \mathit{Position} : \mathit{string};\ \mathit{Salary} : \mathit{nat}\}. \end{array}$$

The Record keyword introduces a new record in Coq. In this case its name is *Payroll*. The types *nat* and *bool* are primitive types in Coq, and *string* is a type we have defined. The *JoinInd* field is the one which indicates whether or not (value *true* or *false*, respectively) the person who owns this data has given permission to use it in a join operation. The *mkPay* constant is used to build individual records. For example, if n, b, s, and m are values of types *nat*, *bool*, *string*, and *nat*, respectively, then the term $(\mathit{mkPay}\ n\ b\ s\ m)$ is a *Payroll* record whose PID value is n, *JoinInd* value is b, etc.

A partial definition of the other two records we use is below.

$$\begin{array}{l} \textsf{Record}\ \mathit{Employee} : \mathit{Set} := \\ \quad \mathit{mkEmp}\ \{\mathit{Name} : \mathit{string};\ \mathit{EID} : \mathit{nat};\ \ldots\}. \\ \textsf{Record}\ \mathit{Combined} : \mathit{Set} := \\ \quad \mathit{mkComb}\ \{\mathit{CID} : \mathit{nat};\ \mathit{CName} : \mathit{string};\ \mathit{CSalary} : \mathit{nat}; \ldots\}. \end{array}$$

The *Employee* record is the one that will be joined with *Payroll*. The PID and EID fields must have the same value and *JoinInd* must have value *true* in order to perform the join. The *Combined* record is the result of the join. The CID field represents the common value of PID and EID. All other fields come from either one or the other record.

In general, how do the different players know the names of the fields in different Ds? Firstly, names of the sensitive fields could be standardized, which in a way is already happening with XML. Alternatively, in a few databases generally relied on, e.g. government health records or driving records, these names would be disclosed to $Veri$. In this example, for simplicity we specify exactly what fields are in each record. We could alternatively express it so that the user's privacy could be ensured independently of the exact form of these records (as long as they both have an ID field, and at least one of them has a *JoinInd* field).

The Definition keyword introduces a definition in Coq. The following defines a function which takes an *Employee* and *Payroll* record and returns the *Combined* record resulting from their join.

$$\begin{array}{l}\textsf{Definition}\ \mathit{mk\_Combined} : \mathit{Employee} \rightarrow \mathit{Payroll} \rightarrow \mathit{Combined} := \\ \quad [E : \mathit{Employee}][P : \mathit{Payroll}] \\ \quad (\mathit{mkComb}\ (\mathit{EID}\ E)\ (\mathit{Name}\ E)\ (\mathit{Salary}\ P)\ \ldots).\end{array}$$

The term $(EID\ E)$ evaluates to the value of the EID field in record E. The CID field of the new record is obtained by taking EID from E, *CName* is obtained by taking *Name* from E, *CSalary* is obtained by taking *Salary* from P, etc.

The main function implementing the join operation is defined in Coq as:

$$\begin{array}{l}\textsf{Fixpoint}\ \mathit{Join}\ [Ps : \mathit{list\ Payroll}] : (\mathit{list\ Employee}) \rightarrow (\mathit{list\ Combined}) := \\ \quad [Es : \mathit{list\ Employee}] \\ \quad \textsf{Cases}\ Ps\ \textsf{of} \\ \quad\ \ \mathit{nil} \qquad\qquad \Rightarrow\ (\mathit{nil\ Combined}) \\ \quad \mid (\mathit{cons}\ p\ ps)\ \Rightarrow\ (\mathit{app}\ (\mathit{check\_JoinInd\_and\_find\_employee\_record}\ p\ Es) \\ \qquad\qquad\qquad\qquad\qquad (\mathit{Join}\ ps\ Es)) \\ \quad \textsf{end}.\end{array}$$

FixPoint indicates a recursive definition. We represent the set of payroll records in the database using the built in datatype for lists in Coq, and similarly for the other sets. *Join* takes lists Ps of payroll records and Es of employee records as arguments, and is defined by case on the structure of Ps using the *Case* syntax presented above. In general, to evaluate the expression

$$Case\ x : M\ of\ M_1 \Rightarrow N_1, \ldots, M_n \Rightarrow N_n$$

the argument x of type M is matched against the patterns $M_1, \ldots, M_n$. If the first one that matches is M_i, then the value N_i is returned. In this example, Ps is either the empty list (*nil*) or the list (*cons p ps*) with head p and rest of the list *ps*. In the first case, an empty list of combined records is returned. In the second case, the function *check_JoinInd_and_find_employee_record* (not shown here) is called. Note that it takes a single *Payroll* record p and the entire list of *Employee* records Es as arguments. It is defined by recursion on Es. If a record in Es is found (1) whose EID matches the PID of p, and (2) whose *JoinInd* field has value *true*, then *mk_Combined* is called to join the two records. A list of length 1 containing this record is returned. Otherwise, an empty list of *Combined* records is returned. Function *app* is Coq's append function used to combine the results of this function call with the recursive call to *Join*.

As stated in the previous section, player C states permissions as a predicate P_C that must hold of programs S. In this example, *Join* is the program S. P_C can be expressed as the following definition where S is the formal parameter:

$$\begin{array}{l}\textsf{Definition}\ Pc := \\ \quad [S : ((\mathit{list\ Payroll}) \rightarrow (\mathit{list\ Employee}) \rightarrow (\mathit{list\ Combined})) \rightarrow \mathit{Prop}] \\ \quad \forall Ps : \mathit{list\ Payroll}.\forall Es : \mathit{list\ Employee}.(\mathit{UniqueJoinInd}\ Ps) \rightarrow \\ \quad \forall P : \mathit{Payroll}.(In\ P\ Ps) \rightarrow ((\mathit{JoinInd}\ P) = \mathit{false}) \rightarrow \\ \quad \neg\exists C : \mathit{Combined}((In\ C\ (S\ Ps\ Es)) \wedge ((CID\ C) = (PID\ P))).\end{array}$$

This predicate states that for any payroll record P with a *JoinInd* field with value *false*, there will be no combined record C in the output of the code S such that the *CID* field of C has a value the same as the *PID* field of P.

The theorem that is written $P_C(S)$ in the previous section is obtained in this case by applying the Coq term *Pc* to *Join* (written (*Pc Join*) in Coq). By replacing the formal paramter S by the actual parameter *Join* and expanding the definition of *Pc*, we obtain the theorem that we have proved in Coq. A request to Coq's proof checking operation to check this proof is thus a request to verify that the preferences of the user are enforced by the *Join* program.

In the theorem, the constant In represents list membership in Coq. The *UniqueJoinInd* predicate is a condition which will be satisfied by any well-formed database with only one payroll record for each PID. We omit its definition.

The proof of (*Pc Join*) proceeds by structural induction on the list Ps. It makes use of seven lemmas, and the whole proof development is roughly 300 lines of Coq script. Compiling this proof script (which includes fully checking it) takes 1 second on a 600MHz Pentium III running linux.

5 Acceptance

In a design of a system which would be used by many different players, close attention needs to be paid to their concerns and interests, lest the system will not be accepted. Firstly, individuals C need to be given an easy tool in which to express their positive and negative permissions. In the design of the permissions language, we are taking into account the kind of data being mined (different Ds), and the schema of processing (joins, different classifiers, etc). Initially, a closed set of permissions could be given to them, from which they would choose their preferences. Such permissions could be encoded either on a person's smart card, or in C's entry in the Public Key Authority directory. More advanced users could use a symbolic language in which to design their permissions. Such a language needs to be designed, containing the typical database and data mining/machine learning operations.

Secondly, who could be the $Veri$ organization? It would need to be a generally trusted body with strong enough IT resources and expertise to use a special-purpose proof checker and perform the verifications on which the scheme proposed here is based. One could see a large consumer's association playing this role. Alternatively, it could be a company which makes its mandate fighting privacy abuses, e.g. Junkbusters.

Thirdly, if the scheme gains wider acceptance, developers of data mining tools can be expected to provide theorems (with proofs) that their software S respects the standard permissions that Cs specify and $Veri$ supports. These theorems and their proofs will be developed in a standard language known by both the developers and $Veri$; we use Coq as the first conceptual prototype of such a language.

Fourthly, what can be done to make organizations involved in data mining (Org in this proposal), and tools providers, accept the proposed scheme? We believe that it would be enough to recruit one large Org and one recognized tool provider to follow the scheme. The fact that, e.g., a large insurance company follows this approach would need to be well publicized in the media. In addition, $Veri$ would grant a special logo, e.g. "Green Data Miner", to any Org certified to follow the scheme. The existence of one large Org that adheres to this proposal would create a subtle but strong social pressure on others to join. Otherwise, the public would be led to believe that Orgs that do not join in fact do not respect privacy of the people whose data they collect and use. This kind of snowball model exists in other domains; it is, e.g., followed by Transparency International.

6 Discussion and Future Work

The paper introduces a new method which implements a mechanism enforcing data ownership by the individuals to whom the data belongs. This is a preliminary description of the proposed idea which, as far as we know, is the first technical solution guaranteeing privacy of data owners understood as their full control over the use of the data, providing verifiable Use Limitation Principle, and supplying a mechanism for opt-in data collection.

The method is based on encoding permissions on the use of the data as theorems about programs that process and mine the data. Theorem proving techniques are then used to express the fact that these programs actually respect the permissions. This initial proposal outlines the method, describes its components, and shows the detailed example of the encoding. We rely on some of the existing tools and techniques for representing the permissions and for checking the theorems about the code that claims to respect them. We also discuss some of the auxiliary techniques needed for the verification.

We are currently working on a prototype of the system described in this paper. This prototype uses some of the Weka's data mining functions as A. We translate the permission-implementing modification of the Weka code into CIC's functional language and build the proof that the CIC code respects the permission stated above. Coq proof checking then automatically checks that the theorem about the modified code is true, which guarantees that the user's constraint is respected by the modified Weka code. Furthermore, we are considering how the experience with Weka could be extended to one of the commercial data mining systems.

A lot of work is left to implement the proposed method in a robust and efficient manner, allowing its wide adoption by data mining tool developers and organizations that perform data mining, as well as by the general public. A permission language acceptable for an average user must be designed and tested. A number of tools assisting and/or automating the activities of different players need to be developed. Firstly, a compiler of the permissions language into the formal (here, CIC) statements is needed. Another tool assisting the translation of live code (e.g. Java) into the formal representation (CIC) must also be developed. Our vision is that with the acceptance of the proposed method such formal representation will become part of the standard documentation of the data mining software. Finally, a tool assisting construction of proofs that programs respect the permissions, and eventually building these proofs automatically, is also needed.

An organization sympathetic to the proposed approach and willing to implement and deploy it on a prototype basis needs to be found. This Org will not only protect the owners of the data, but can also act as a for-profit data provider. The latter aspect is possible as the proposed method supports an opt-in approach to data collection, based on the user's explicit consent. A commercial mechanism rewarding the opting-in individuals could be worked out by this organization and tested in practice.

Acknowledgements. The authors acknowledge the support of the Natural Sciences and Engineering Research Council of Canada, Computing and Information Technologies Ontario, and the Centre National de la Recherche Scientifique (France). Rob Holte, Francesco Bergadano, Doug Howe, Wladimir Sachs, and Nathalie Japkowicz are thanked for discussing some aspects of the work with us.

References

[1] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–255. ACM, May 2001.
[2] R. Agrawal and R. Srikant. Privacy-preserving data mining. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *2000 ACM SIGMOD International Conference on Management of Data*, pages 439–450. ACM, May 2000.
[3] G. Barthe, G. Dufay, L. Jakubiec, B. Serpette, and S. Sousa. A formal executable semantics of the JavaCard platform. In *European Symposium on Programming*, pages 302–319. Springer-Verlag, 2001.
[4] Y. Bertot. Formalizing a JVML verifier for initialization in a theorem prover. In *Computer-Aided Verification*, pages 14–24. Springer-Verlag, 2001.
[5] Electronic Privacy Information Center and Junkbusters. Pretty poor privacy: An assessment of P3P and internet privacy. http://www.epic.org/reports/prettypoorprivacy.html, June 2000.
[6] K. Coyle. P3P:pretty poor privacy?: A social analysis of the platform for privacy preferences (P3P). http://www.kcoyle.net/p3p.html, June 1999.
[7] Information and Privacy Commissioner/Ontario. Data mining: Staking a claim on your privacy. http://www.ipc.on.ca/english/pubpres/papers/datamine.htm#Examples, January 1998.
[8] D. G. Ries. Protecting consumer online privacy — an overview. http://www.pbi.org/Goodies/privacy/privacy_ries.htm, May 2001.
[9] R. L. Rivest. RFC 1321: The MD5 message-digest algorithm. Internet Activities Board, 1992.
[10] The Coq Development Team. The Coq Proof Assistant reference manual: Version 7.2. Technical report, INRIA, 2002.
[11] W3C. Platform for privacy preferences. http://www.w3.org/P3P/introduction.html, 1999.