Privacy and Utility of Inference Control Mechanisms for Social Computing Applications

Seyed Hossein Ahmadinejad
Nulli
shan@nulli.com

Philip W.L. Fong Reihaneh Safavi-Naini
University of Calgary
{pwlfong, rei}@ucalgary.ca

ABSTRACT

Modern social computing platforms (e.g., Facebook) are extensible. Third-party developers deploy extensions (e.g., Facebook applications) that augment the functionalities of the underlying platforms. Previous work demonstrated that permission-based protection mechanisms, adopted to control access to users' personal information, fail to control inference — the inference of private information from public information. We envision an alternative protection model in which user profiles undergo sanitizing transformations before being released to third-party applications. Each transformation specifies an alternative ***view*** of the user profile. Unlike permission-based protection, this framework addresses the need for inference control.

This work lays the theoretical foundation for view-based protection in three ways. First, existing work in privacy-preserving data publishing focuses on structured data (e.g., tables), but user profiles are semi-structured (e.g., trees). In information-theoretic terms, we define privacy and utility goals that can be applied to semi-structured data. Our notions of privacy and utility are highly targeted, mirroring the set up of social computing platforms, in which users specify their privacy preferences and third-party applications focus their accesses on selected components of the user profile. Second, we define an algebra of trees in which sanitizing transformations previously designed for structured data (e.g., generalization, noise introduction, etc) are now formulated for semi-structured data in terms of tree operations. Third, we evaluate the usefulness of our model by illustrating how the privacy enhancement and utility preservation effects of a view (a sanitizing transformation) can be formally and quantitatively assessed in our model. To the best of our knowledge, ours is the first work to articulate precise privacy and utility goals of inference control mechanisms for third-party applications in social computing platforms.

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China

DOI: http://dx.doi.org/10.1145/2897845.2897878

CCS Concepts

•**Security and privacy → Security requirements; Social network security and privacy;**

Keywords

Social computing; Facebook applications; inference attack; privacy; utility; view-based protection; sanitizing transformation; verification; composition

1. INTRODUCTION

Today's Social Network Systems (SNSs) are designed such that their functionalities can be extended by third-party extensions (e.g., Facebook applications). Extensions can access user data stored in the SNS through an SNS Application Programming Interface (API). To protect user privacy, a user can explicitly specify what she wants to hide from or share with an extension. The SNS API requires the extension to seek permissions from the user. Access is allowed only if the corresponding permission is granted by the user.

SNS API Inference Attacks. The inadequacy of permission-based protection for users of SNS extensions was first articulated by Ahmadinejad *et al.* [3], through the following example. Suppose a user does not want to share her birthday with a Facebook application, but she is willing to grant the application access to her "wall," for such accesses are needed by the application to deliver its functionalities. She then sets up the permissions to reflect the above privacy preference: grant permission to access her wall, deny permission to access her birthday. What she may not be aware of is that a malicious application may scan through her wall, looking for a day in the year in which there is a high number of postings with "happy birthday" wishes, thereby inferring her birthday, even though the latter has been made inaccessible.

This type of information leakage, that is, inferring users' sensitive information from their accessible information through an SNS API, is called ***SNS API inference attacks***. This type of attacks was first introduced in [3] (note that inference attacks in databases were firstly introduced in [20], but our focus is on SNS API inference attacks). The feasibility and accuracy of sample inference algorithms were empirically evaluated in subsequent works [4, 5]. Success rates of the inference algorithms were found to be alarmingly high. It was also shown that SNS API inference attacks can be employed as a building block for launching further security attacks, including, for example, phishing and bypassing authentication challenges.

View-based Protection of User Data. The above example illustrates an important point: controlling access cannot control inference. Privacy is not simply about access authorization, but about breaking data correlation so that inference becomes impossible.

In this work, we envision an alternative protection framework in which user profiles undergo sanitizing transformations before being released to third-party applications. Each transformation specifies an alternative representation of the user profile that we call a ***view***. A privacy policy is a specification of what transformations must be applied to the user profile prior to its disclosure to various applications. We call such a protection framework ***view-based protection***. Unlike permission-based protection, this framework can address the need for inference control. A transformation perturbs the statistical correlation of data, and thus upsets the attacker's ability to infer sensitive information.

Contributions. This work lays the theoretical foundation for view-based protection. Specifically, we claim the following contributions:

1. Existing work in privacy-preserving data publishing focuses on structured data (e.g., tables) [1], but user profiles are semi-structured (e.g., trees). In information-theoretic terms, we formulated privacy and utility goals that can accommodate semi-structured data, in which "attributes" are defined via complex queries that locate, extract or even combine information dispersed in various parts of the semi-structured data (§3). Our notions of privacy and utility are highly targeted (§3.2), mirroring the set-up of social computing platforms, in which users specify their privacy preferences (what information needs protection) and third-party applications declare their accesses on selected components of the user profile (what information needs to be available). We identified the conditions under which sanitizing transformations are safely composable, such that the privacy enhancing and utility preserving effects of successive transformations are accumulative (§3.3, §3.5, §3.6). We formally articulated when there will be an inevitable trade-off between privacy and utility, and identified conditions under which a sanitizing transformation performs the trade-off in a productive manner (§3.4).
2. Suppression, generalization, permutation and noise introduction are common sanitizing transformations originally designed for structured data. We defined an algebra of trees through which analogues of these four transformations are formulated for semi-structured data (§4). The privacy enhancing and utility preserving characteristics of these four transformations are formally articulated (§5).
3. We evaluated our model in two ways (§6). First, we illustrated how the privacy enhancing and utility preserving effects of a view can be formally and quantitatively assessed in our model (§6.1). Second, we used our model to analyze the relative merits of different sanitizing strategies on semi-structured data (§6.2).

To the best of our knowledge, ours is the first work to articulate precise privacy and utility goals of inference control for third-party applications in social computing platforms.

2. VIEW-BASED PROTECTION

This section gives an overview of view-based protection, and motivates our methodology by drawing an analogy between the goal of this work and that of program verification.

Protection via Views. We assume every user is the administrator of her own profile. Her responsibility as an administrator is to specify, for each third-party application (or each category of applications), what information is considered sensitive and thus requires protection. Note that this is a specification of ***privacy preference***, and not permissions.

We assume every third-party application will explicitly declare what information in the user profile it plans to consume in order to deliver its functionalities. Again, this is not a request for permissions, but rather a ***utility declaration***.

Intuitively, the privacy goal is to break the correlation between the information declared to be sensitive by the privacy preference, and the information in the published user profile. In that way, when the application accesses the published user profile, it cannot infer the sensitive information.

This privacy goal is achieved in view-based protection not by denying access to the sensitive information. Instead, a sanitizing transformation is applied to the user profile before the latter is made available to the third-party application. The transformed profile is called a ***view*** of the original profile. When the third-party application makes a query against the user profile, the query is evaluated over the transformed profile rather than the original profile. A properly designed transformation is supposed to break the correlation between the sensitive information and the transformed profile.

The transformation, however, may destroy the usefulness of profile information. The utility goal, intuitively stated, is to preserve as much as possible the availability of useful information as specified in the utility declaration.

Who is responsible for engineering a view is a matter of platform design. One possibility is to have dedicated third-party developers engineer primitive transformations, with formally certified effects on privacy enhancement and utility preservation. These primitive transformations can become building blocks for views. The users, under the guidance of the social computing platform and the advice of the application developer, can then compose a view out of these building blocks. For this protection scheme to be viable, there shall be formal means for verifying if the view that is composed out of primitive transformations does indeed fulfill both the privacy preference of the user and the utility declaration of the application.

Formal Verification of Views. The task of verifying views is analogous to the formal verification of program correctness, which aims at providing formal guarantees that the program behaves according to specification. Program correctness is usually established in a compositional manner. For example, in Floyd-Hoare Logic [12], a program is specified in terms of a precondition and a postcondition: if the precondition is satisfied prior to program execution, the postcondition shall be met when the program terminates. The semantics of an individual program statement is specified in terms of an inference rule, which delineates the precondition and postcondition of that statement. A proof of correctness is obtained by composing the inference rules of the statements in the program, and thus inferring that the program postcondition follows from the program precondition.

In this work, we propose a theoretical framework for verifying that a view meets its intended privacy and utility

goals. This proposal is analogous to the compositional nature of program verification. We envision that a view is composed of more primitive transformations. We articulate the conditions under which the privacy enhancing and utility preserving effects of individual transformations are accumulative. Under such conditions, composition of transformations is safe, and we can verify a view by verifying each building block in turn. These conditions are captured in the form of an inference rule. The inference rule are then instantiated for various sanitization strategies, including suppression, generalization, permutation and noise introduction.

3. PRIVACY AND UTILITY GOALS

This section develops a framework for assessing if views composed of primitive transformations meet quantitative goals of privacy and utility. The framework is general enough to accommodate either structured or semi-structured data.

3.1 Preliminaries

We begin with some definitions which will be used throughout the paper.

We write D_A for the domain of a random variable A.

DEFINITION 1 (CONDITIONAL INDEPENDENCE [19]). *Given random variables A, B and C, variables A and C are said to be* **conditionally independent given** *B iff:*

$$\forall a \in D_A, b \in D_B, c \in D_C \,.\; \Pr(A = a \mid B = b, C = c) = \Pr(A = a \mid B = b)$$

In other words, when B is known, learning C does not change our knowledge of A. In such a case, we write $(A \perp\!\!\!\perp C) \mid B$.

Bayesian networks, which are probabilistic graphical models, can help in showing the dependency between random variables. Bayesian networks are directed acyclic graphs where nodes are random variable, and an edge from one random variable to another indicates that the former causes the latter. In this work, we use Bayesian networks to depict the dependencies among random variables.

DEFINITION 2 (SHANNON ENTROPY [10]). *The* Shannon entropy $H(A)$ *of a random variable A is defined as follows:*

$$H(A) = -\sum_{a \in D_A} \Pr(A = a) \cdot \log \Pr(A = a)$$

Intuitively, $H(A)$ measures the uncertainty about A.

DEFINITION 3. *Given another random variable B, the* conditional entropy $H(A \mid B)$ *is defined by*

$$H(A \mid B) = \sum_{b \in D_B} \Pr(B = b) \cdot H(A \mid B = b)$$

If A is a deterministic function of B, then $H(A \mid B) = 0$.

DEFINITION 4. *The* mutual information *between A and B, which is the amount of information shared between A and B is defined by*

$$I(A; B) = I(B; A) = H(A) - H(A \mid B) = H(B) - H(B \mid A)$$

PROPOSITION 1 (DATA PROCESSING INEQUALITY [10]). *Given random variables A, B, and C:*

$$(A \perp\!\!\!\perp C \mid B) \Rightarrow I(A; B) \geq I(A; C)$$

3.2 Sensitive and Useful Information

Suppose $\mathcal{A}$ is the set of all user profiles. The information accessible via a profile is a random variable A where D_A, the domain of A, is $\mathcal{A}$. A sanitizing transformation of $a \in \mathcal{A}$ is a (deterministic or probabilistic) function $t : \mathcal{A} \to \mathcal{A}$. Furthermore, a transformation t induces a random variable $t(A)$ with distribution $P(t(A) \mid A)$.

The user specifies in her privacy preference (§2) what components of her profile are considered sensitive. We model this specification as a deterministic function $s : \mathcal{A} \to \mathcal{A}$ that extracts those components from the profile. Now s induces a random variable $S = s(A)$. Assuming s to be deterministic is reasonable because, even if S is not originally a deterministic function of A, a rational adversary always infers S to have the value that maximizes $P(S \mid A = a)$ for a given a. This is the best strategy an adversary can adopt.

An SNS extension delivers functionalities to the user by extracting certain information from the user profile A. In practice, the extension specifies in its utility declaration (§2) the components of the user profile it needs to query in order to deliver its functionalities. We model this declaration as a deterministic function $u : \mathcal{A} \to \mathcal{A}$ that extracts the useful components from the profile. Now u induces a random variable $U = u(A)$. The degree in which information concerning U is preserved after the user profile undergoes sanitizing transformation, is the utility of the sanitized profile.

The privacy and utility goals are now clear. A transformation t is to be devised such that the correlation between $t(A)$ and S is minimized (privacy), and the correlation between $t(A)$ and U is maximized (utility). This implies the need for a ***correlation measure***. Let X_Y be a correlation measure between random variables X and Y. The privacy and utility goals are then respectively minimizing $S_{t(A)}$ and maximizing $U_{t(A)}$.

Instantiating the correlation measures. The privacy and utility goals can be instantiated by different choices of correlation measure. Two possibilities are given below:

1. ***Shannon entropy:*** Take X_Y to be the mutual information $I(X; Y)$ between X and Y, i.e., $H(X) - H(X \mid Y)$.
2. ***Min-entropy/vulnerability [22]:*** Define X_Y to be the following quantity:

$$H_\infty(X) - H_\infty(X \mid Y) = \log \frac{V(X \mid Y)}{V(X)}$$

It has been shown in the literature that min-entropy is a stronger measure compared to Shannon entropy and mutual information [22]. However, sometimes a less restrictive and less demanding notion is more helpful in designing constructions and their analysis. For that reason, we adopt the notion of mutual information in this work as our correlation measure. Instantiating the correlation measure with min-entropy is left to future work.

With the above instantiation, the amount of information disclosed by A regarding sensitive information S, called the ***information leakage*** of A about S, is modelled by

$$S_A = I(S; A) \tag{1}$$

Moreover, the utility of released information A w.r.t. some useful information U is measured by

$$U_A = I(U; A) \tag{2}$$

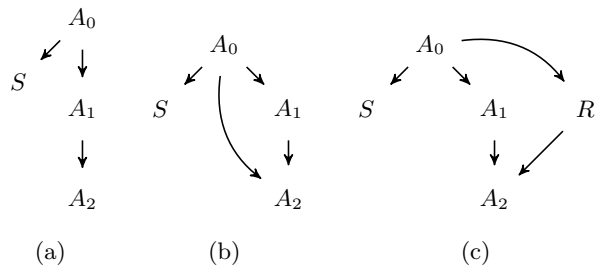

Figure 1: Sample Bayesian networks

Information leakage ranges between $H(S)$ and 0, while utility ranges between $H(U)$ and 0.

3.3 Modular Development of Transformations

Typically, the transformation t is engineered in a modular manner. That is, t is the composition of a number of primitive transformations, that is, $t = t_n \circ \ldots \circ t_2 \circ t_1$ so that:

$$A_0 = A \text{ and } A_i = t_i(A_{i-1}) \text{ for } 1 \leq i \leq n$$

The profile to be released to the SNS extension is therefore

$$t(A) = A_n = t_n(\ldots t_2(t_1(A)))$$

In the following, we characterize the conditions under which the composition above has predictable effects.

3.3.1 What Could Go Wrong

Modular development presumes that each component transformation further reduces the correlation between the published profile and the sensitive information: i.e., $I(S; A_i) \leq I(S; A_{i-1})$. If the behaviour of a component transformation t_i depends on the original profile (A_0), or it looks up some publicly available data sources that correlate with S, we can no longer guarantee the above assumption.

To illustrate the above, consider a profile A_0, which is transformed first into A_1 and then into A_2. Let the Bayesian network in Fig. 1a show the dependencies among the random variables. Clearly, A_1 has no more mutual information with S than A_0 does, and A_2 has no more mutual information with S than A_0 and A_1 do. So in each of the two rounds of transformation, the amount of information that could be inferred for S has been reduced.

Now consider the situation in Fig. 1b, in which A_2 depends not only on A_1 but also A_0. This may be because transformation t_1, after reading A_0, leaves some state information behind that t_2 consumes. In that case, A_2 depends also on A_0. Now, both $I(S; A_1)$ and $I(S; A_2)$ are no larger than $I(S; A_0)$, but we can no longer claim that $I(S; A_2) \leq I(S; A_1)$. In other words, although transforming A_0 to A_2 in two rounds does not reveal more information about S than prior to the two transformations, the second transformation may reverse some of the privacy enhancing effect of the first transformation: i.e., t_2 leaks some information to A_2 that is not contained in A_1.

A similar anomaly occurs in the situation of Fig. 1c. Suppose t_2 is a probabilistic function. The source of randomness is captured by a random variable R. Suppose R is biased, and the bias correlates with A_0. Now we can no longer guarantee $I(S; A_2) \leq I(S; A_1)$. Note that if R was not connected to A_0, then such a guarantee would hold.

Rule: *Universal Inference Rule*

1. A : either the original user profile (A_0), or a partially sanitized version of the original user profile (A_i)
2. $t : D_A \rightarrow D_A$: a sanitizing transformation (either deterministic or probabilistic)
3. S : a deterministic function of the original user profile (A_0), modelling the sensitive information to be protected
4. U : a deterministic function of the original user profile (A_0), modelling information to be consumed by the SNS extension

Premises:

1) $S \perp\!\!\!\perp t(A) \mid A$.
2) $U \perp\!\!\!\perp t(A) \mid A$.

Conclusions:

1) $0 \leq S_{t(A)} \leq S_A$.
2) $0 \leq U_{t(A)} \leq U_A$.

Table 1: Universal Inference Rule

3.3.2 Safe Composition

We articulate the conditions under which sanitizing transformations are composable in the form of an inference rule in Table 1. The inference rule specifies a set of premises and a set of conclusions. The conclusions are guaranteed to hold if the premises are established. The rule captures the privacy enhancing and utility preserving effects of some component transformation t (i.e., one of the t_i in the previous discussion), with input A being either the original user profile (A_0) or a partially sanitized version of the original user profile (A_i). The sensitive information S and the utility U may or may not be deterministic functions of input A, depending on whether A is the original user profile (A_0).

Premises. The two premises assert the conditional independence of $t(A)$ respectively with S and U given the input A. In other words, the information of S and U that is contained in $t(A)$ originates solely from A. This essentially rules out the anomalies in §3.3.1.

Conclusions. The conclusions of the inference rule express the effects of transformation t on (1) controlling the inference of sensitive information (privacy enhancement) and (2) reducing the usefulness of the published profile (utility degradation).

1. *Privacy enhancement.* The key of conclusion 1 is an upper bound for $S_t(A)$, that is $I(S; t(A)) \leq I(S; A)$. This shows that the correlation between the sensitive information and the user profile after the transformation is no stronger than that before the transformation.
2. *Utility degradation.* Similarly, the upper bound of conclusion 2 amounts to $I(U; t(A)) \leq I(U; A)$. This means transformation reduced the utility of the profile. The best case scenario is when $I(U; A) = I(U; t(A))$ (i.e., perfect preservation of utility), while the worst case scenario is when $I(U; t(A)) = 0$ (i.e., utility is completely destroyed).

Composition. Suppose $t = t_n \circ \ldots t_2 \circ t_1$, and each t_i satisfies the premises of the inference rule. Then we know each round of transformation makes progress in reducing the correlation of the user profile with the sensitive information. That is,

no leakage of information as discussed in §3.3.1 would inject sensitive information back into the user profile.

Soundness. The soundness of the inference rule follows immediately from the Data Processing Inequality (Proposition 1). Since $t(A)$ and S are conditionally independent given A, $I(S;t(A)) \leq I(S;A)$ holds, which in turn implies conclusion 1. Similarly, the second premise leads to the second conclusion (regarding utility). Note that if we remove $(S \perp\!\!\!\perp t(A) \mid A)$ from the premises, the conclusion $H(S \mid A) \leq H(S \mid t(A))$ would still be guaranteed in the special case of $H(S \mid A) = 0$. The similar holds for utility.

3.4 Trading Off Privacy & Utility

We noted above that a transformation may fail to perfectly preserve utility ($I(U;t(A)) < I(U;A)$). The reason lies in the fact that S and U have opposite roles in the inference problem. When we transform A, on the one hand we aim at destroying the correlation between A and S, and on the other hand, we want to preserve the correlation between A and U. This suggests that we have to pursue a trade-off between hiding S and revealing U.

3.4.1 The Inevitable Trade Off

Suppose there is no mutual information between S and U: i.e., $I(S;U) = 0$. Theoretically, it is possible to find a transformation t such that $I(t(A);U) = I(A;U)$ and $I(t(A);S) = 0$. In intuitive terms, if the information in A that is used for inferring S is completely independent of the information in A that is used for inferring U, then it may be possible to construct a transformation to sanitize A such that the sanitized A contains the same amount of information about U as it used to have, but all traces of information regarding S has been eliminated. For example, consider the situation where $I(A;U) = H(U)$. If we choose $t(A)$ to be equal to U, then, because $I(S;U) = 0$, $I(t(A);S) = 0$ holds. Moreover, $I(U;U) = H(U)$ also holds. In such a case, we can guarantee utility is completely preserved and that $t(A)$ does not reveal anything about S.

Now assume there is some mutual information between S and U. In that case, the complete protection of sensitive information and the perfect preservation of utility cannot be achieved simultaneously. This is captured by the following:

PROPOSITION 2. *Suppose there is some mutual information between S and U (i.e., $I(S;U) > 0$). Suppose further that U is a deterministic function of A (i.e., $H(U \mid A) = 0$). Then it is not possible to find a transformation t satisfying both of the following two conditions:*

1. *$t(A)$ does not leak information about S: i.e., $I(S;t(A)) = 0$, and*
2. *$t(A)$ perfectly preserves information about U: i.e., $I(U;t(A)) = I(U;A)$.*

A proof of the above result is given in [2, Proposition 4.4.1].

In other words, if S and U share mutual information, and U is a deterministic function of A, then it is impossible to find a transformation t such that (a) $t(A)$ does not carry any information about S, and (b) $t(A)$ contains as much information about U as A does. That is, either information leakage or utility degradation is inevitable.

3.4.2 When Compromise is Necessary

In summary, if $I(S;U) > 0$, one of the following three conditions must hold: *1*) some information leakage (i.e., $S_{t(A)} > 0$) but no utility degradation (i.e., $U_{t(A)} = U_A$), *2*) some utility degradation (i.e., $U_{t(A)} < U_A$) but no information leakage (i.e., $S_{t(A)} = 0$), or *3*) both information leakage and utility degradation. When either information leakage or utility degradation is inevitable, we want to ensure that if some information of S is leaked, the leaked information contributes to utility (i.e., the leaked information concerns knowledge of U). Otherwise, such information does not need to be released as it only gives away information about S with no positive effect on utility preservation — we call this ***unnecessary information leakage***.

Likewise, suppose we accept utility degradation to be inevitable, and thus we refrain from releasing certain information about U. It is reasonable for us to expect that such information regarding U must contain information of S as well. Otherwise, we end up eliminating information of U without contributing to privacy enhancement — we call this ***unnecessary information transformation***.

When there is mutual information between U and S, compromising between preserving utility and protecting sensitive information is necessary. However, this compromise must be done for a legitimate reason. We shall avoid unnecessary information transformation and unnecessary information leakage.

3.4.3 Unnecessary Information Leakage

Unnecessary information leakage occurs when $t(A)$ contains some information of S that is not contained in U. Formally, this means:

$$H(S \mid t(A), U) < H(S \mid U), \text{ that is, } I(S;t(A) \mid U) > 0$$

As a result, in order to avoid unnecessary information leakage, (3) must hold:

$$H(S|U) = H(S|t(A), U), \text{ that is, } I(S;t(A) \mid U) = 0 \quad (3)$$

3.4.4 Unnecessary Information Transformation

Unnecessary information transformation occurs when, even by knowing S and $t(A)$ together, U cannot be inferred without uncertainty. Formally, this means the condition below:

$$H(U \mid t(A), S) > 0, \text{ that is, } I(U;t(A), S) < H(U)$$

As a result, in order to avoid unnecessary information transformation, we require that (4) holds:

$$H(U \mid t(A), S) = 0, \text{ that is, } H(U) = I(U;t(A), S) \quad (4)$$

Note that this interpretation of unnecessary information transformation considers utility as the primary factor in determining when transformation is overdone. It is required that, if utility needs to be sacrificed, it should be because of protecting sensitive information.

3.5 Measuring Privacy & Utility

According to the previous section, a transformation that satisfies (3) and (4) guarantees that there is no unnecessary information leakage and no unnecessary information transformation. But there might be multiple transformations of A that meet those conditions. This motivates the need for quantifying the amount of privacy enhancement and utility degradation caused by a transformation. Such measurements enables us to compare transformations.

3.5.1 Privacy Enhancement Measure

Given a transformation t, we define a measure called the ***privacy enhancement*** of t as follows:

$$S_{t,A} = S_A - S_{t(A)} \tag{5}$$

$S_{t,A}$ measures how successful t is in transforming A for the purpose of protecting S. $S_{t,A}$ is always no smaller than 0 and no larger than S_A. Note that we assume t is a function that satisfies the premises of our universal inference rule.

PROPOSITION 3. *If t does not result in any unnecessary information leakage, the maximum amount of information leakage of $t(A)$ about S would be $I(S;U)$.*

A proof of the above result is given in [2, Proposition 4.4.2]. Note that if $I(S;U) = 0$, then $S_{t(A)} = 0$ too, which means $t(A)$ does not contain any information about S.

3.5.2 Utility Degradation Measure

In order to measure how good a transformation t performs in terms of preserving the usefulness of data, we define a measure called ***utility degradation*** as follows:

$$U_{t,A} = U_A - U_{t(A)} \tag{6}$$

PROPOSITION 4. *If t does not result in any unnecessary information transformation, the maximum amount of utility degradation caused by t would be $I(S;U)$.*

A proof of the above result is given in [2, Proposition 4.4.3].

In summary, the last two sections showed if t causes no unnecessary information transformation and no unnecessary information leakage, the amount of information $t(A)$ discloses about S and the amount of usefulness we lose for U by transforming A to $t(A)$ will both be bounded by $I(S;U)$.

3.6 Revisiting the Universal Inference Rule

In §3.3.2, we formulated a universal inference rule. By introducing the measures of privacy enhancement and utility degradation, we can now restate the universal inference rule by replacing the conclusions in Table 1 with the followings:

$$0 \le S_{t,A} \le S_A \qquad 0 \le U_{t,A} \le U_A$$

The soundness of this restated inference rule follows from the propositions proven in the previous sections. The significance of this restatement of the universal inference rule is that privacy enhancement is additive. That is, if $t = t_n \circ \ldots \circ t_2 \circ t_1$, $A_0 = A$, and $A_i = t_i(A_{i-1})$ for $1 \le i \le n$, then $S_{t,A} = \Sigma_{i=1}^{n} S_{t_i,A_{i-1}}$. Additivity follows from (5). Similarly, utility degradation is also additive (follows from (6)): $U_{t,A} = \Sigma_{i=1}^{n} U_{t_i,A_{i-1}}$.

Based on what we discussed regarding transformation with no unnecessary information leakage and no unnecessary information transformation, we state an optimal universal inference rule in Table 2. The proof of soundness for this rule follows immediately from the propositions we provided in earlier sections. Note that if $I(S;U)$ equals 0, then $S_{t(A)}$ and $U_{t,A}$ both equal 0 which is the best case scenario.

4. A DATA MODEL FOR USER PROFILES

To demonstrate how a transformation (view) is created, we need to define an algebraic structure for modelling and manipulating user profiles for the purpose of sanitization. An algebraic structure has a carrier set containing objects.

Rule: *Optimal Universal Inference Rule*
Premises:
1) $S \perp\!\!\!\perp t(A) \mid A$.
2) $U \perp\!\!\!\perp t(A) \mid A$.
3) (3) holds: no unnecessary info. leakage.
4) (4) holds: no unnecessary info. transformation.
Conclusions:
1) $0 \le S_{t(A)} \le I(S;U)$.
2) $0 \le U_{t,A} \le I(S;U)$.

Table 2: Optimal Universal Inference Rule

A signature with its function symbols is required for constructing objects in the carrier set. An algebraic structure also has a set of predicate symbols, or operators, defined on the carrier set. To model user profiles, a data model is required. Although our framework is not generally limited to any specific data model, we prefer a data model that on the one hand, suits our needs, and on the other hand, simplifies the follow-up discussions. We tend to ignore concerns regarding the implementation of our data model, which is an abstract model proposed solely for the purpose of defining our framework. In practice, the data model could be defined in a different way. Overall, our framework does not impose any constraint on the data model.

4.1 User Profiles as Trees

4.1.1 Rationale

User profiles are semi-structured data. In this work we represent user profiles as trees. We justify this choice of our data model in the following.

Making explicit the hierarchical structure of user data. The information that a user has in her profile naturally forms a hierarchy. For example, there is a hierarchical relation between a photo album and its photos.

Protection significance. The hierarchical structure of profiles typically leads to hierarchical relationships between the accessibility of various data components. For instance, a user might specify that photos uploaded to a photo album inherit the accessibility of the album. Hence, the hierarchical structure of data actually carries protection significance.

Well-established technologies for processing trees. Tree structures are popular for information storage/transportation. For example, XML is a widely used data transport format. The World Wide Web Consortium (W3C) has established standards for a rich set of technologies that interoperate with XML. XML Path Language (XPath) [14] is recommended by W3C to be used for navigating XML documents. Extensible Stylesheet Language Transformations (XSLT) [8] is used for specifying transformations of XML documents. Mature implementations are available for these standards.

4.1.2 Trees as Terms in a Free Algebra

We represent user profiles as terms from a free algebra generated by a domain-dependent signature of function symbols. In particular, a signature Σ is a set of function symbols. Each symbol $f \in \Sigma$ has an arity $arity(f)$. A symbol c for which $arity(c) = 0$ is called a constant symbol. The legitimate terms in the free algebra are the following: 1) every constant symbol c is a term, and 2) if $t_1, \ldots, t_k$ are terms, and f is a function symbol with arity k, then $f(t_1, \ldots, t_k)$

```
profile(basic_info(mike, male, mike@y.com),
   entry(1/2/2013, hello,
      entry(3/4/2013, happy_bd, nil)))
```

Table 3: A sample user profile

is a term. There are no other terms in the free algebra. Hereafter we use "terms" and "trees" interchangeably.

We illustrate how a user profile is modelled as a term. Suppose a profile contains two categories of information: (a) basic information, and (b) a wall. Basic information consists of the user's name, gender, and email address. The wall consists of a number of entries, each of which has a timestamp (indicating the time when the entry was posted), and a message (indicating the text message of the entry). Such a profile can be represented by terms with the following context-free grammar:

Profile ::= `profile(`*Basic_info*`,` *Wall_entry*`)`
Basic_info ::= `basic_info(`*Name*`,` *Gender*`,` *Email*`)`
Wall_entry ::= `entry(`*Timestamp*`,` *Message*`,` *Wall_entry*`)`
| `nil`

The root of the profile is identified by the function symbol `profile`, which has an arity of 2. Nonterminals *Name*, *Gender*, *Email*, *Timestamp* and *Message* expand to constant symbols of the corresponding types. Table 3 illustrates a sample user profile of a male user, Mike, with email address `mike@y.com` who has two entries on his wall. Occasionally, one may want to define custom data types for representing certain components of the user profile. For example, the entries on the wall should be unordered (i.e., a multiset) rather than ordered (i.e., a sequence). To achieve this effect, one can augment the term algebra by an appropriately chosen ***equational theory***. For instance, the following equation may be asserted as an axiom to render entries "commute" with one another, thereby eliminating the significance of ordering among wall entries.

$$\texttt{entry(}TS_1\texttt{,}\ M_1\texttt{,}\ \texttt{entry(}TS_2\texttt{,}\ M_2\texttt{,}\ W\texttt{))} = \texttt{entry(}TS_2\texttt{,}\ M_2\texttt{,}\ \texttt{entry(}TS_1\texttt{,}\ M_1\texttt{,}\ W\texttt{))} \quad (7)$$

It is assumed that an appropriate equational theory is specified for the term algebra to capture properties of various custom data types. For example, a commutative and associative binary function symbol can be used for representing multisets, and a commutative, associative and idempotent binary function symbol can be used for representing sets. When that is the case, for two given profiles $a_1, a_2 \in \mathcal{A}$ where $a_1 = a_2$ and a transformation t, $t(a_1) = t(a_2)$ holds. This means the transformation t is aware of the semantics of data types as induced by the equational theory.

4.2 An Algebra of Trees

4.2.1 Positions and Branches

In order to define algebraic operators for transforming a tree (e.g., by adding or removing nodes), we need to be able to reference different parts of the tree. To that end, we assign a unique address, called a ***position***, to every node in the tree. Formally, a position is a (possibly empty) sequence of positive integers: i.e., a position is a member of $\mathbb{N}^*$, where $\mathbb{N} = \{1, 2, \ldots\}$. The position of the root of the tree is the empty sequence ϵ. Suppose a node $f(a_1, \ldots, a_k)$ has position p, then its children $a_1, \ldots, a_k$ have positions $p \cdot 1, \ldots, p \cdot k$ respectively. For example in Table 3, the position of the function symbol `hello` is $2 \cdot 2$. We inductively define the set $\mathcal{P}os : \mathcal{A} \rightarrow 2^{\mathbb{N}^*}$ of legitimate positions for every tree $a \in \mathcal{A}$ such that (a) if a is a leaf (i.e., constant symbol), then $\mathcal{P}os(a) = \{\epsilon\}$, and (b) if a is of the form $f(a_1, \ldots, a_k)$, where $k \geq 1$, then $\mathcal{P}os(a) = \{\epsilon\} \cup \bigcup_{i=1}^{k} \{i \cdot p \mid p \in \mathcal{P}os(a_i)\}$.

For instance, for the tree a shown in Table 3, $\mathcal{P}os(a) = \{\epsilon, 1, 2, 1 \cdot 1, 1 \cdot 2, 1 \cdot 3, 2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 3 \cdot 1, 2 \cdot 3 \cdot 2, 2 \cdot 3 \cdot 3\}$.

We write $p_1 \leq p_2$ whenever p_1 is a prefix of p_2: i.e., there exists $q \in \mathbb{N}^*$ such that $p_2 = p_1 \cdot q$. We write $p_1 < p_2$ if $p_1 \leq p_2$ but $p_1 \neq p_2$. If neither $p_1 \leq p_2$ nor $p_2 \leq p_1$, then p_1 and p_2 are said to be ***incomparable***. A position $p \in \mathcal{P}os(a)$ is an ***internal position*** if there exists $p' \in \mathcal{P}os(a)$ for which $p < p'$. Otherwise, p is a ***leaf position*** of a.

The tree rooted at a given position of a tree is called a ***branch*** of the latter tree. (What we call a "branch" is typically called a "subtree" in the literature. We reserve the term "subtree" for another purpose — see below.) Operator $\downarrow : \mathcal{A} \times \mathbb{N}^* \rightarrow \mathcal{A}$ is defined such that for a given tree a and every position $p \in \mathcal{P}os(a)$, $a \downarrow p$ extracts the branch rooted at position p in a. More specifically, $a \downarrow \epsilon = a$, and, for $1 \leq i \leq k$, $f(a_1, \ldots, a_k) \downarrow (i \cdot p) = a_i \downarrow p$.

4.2.2 Subtrees and Reduction

Transforming a profile typically starts with identifying parts of the profile that need changing. The rest of the profile will be released as is. Sometimes, only the invariant parts are released, while for other transformations, a modified version of the variant parts are released along with the invariant parts. For example, consider the following profile:

```
profile(basic_info(...), entry(...))        (8)
```

Suppose the variant part of the profile above is the basic information, and the rest is invariant. We identify the invariant parts of the profile by the following tree:

```
profile(ϵ, entry(...))                      (9)
```

Here, the special constant symbol ϵ behaves like a placeholder: ϵ indicates that a branch used to occupy that position, but has been temporarily removed. That placeholder can be filled by a different branch (e.g., obtained by sanitizing the basic information). The profile in (9) is said to be a reduced version of the profile in (8). Given a tree a and a position $p \in \mathcal{P}os(a)$, p is said to be an ***empty position*** of a if $a \downarrow p = \epsilon$, or a ***filled position*** otherwise.

We define a binary relation ($\sqsubseteq$) over trees in $\mathcal{A}$ to indicate if a tree is the reduced version of another. Given $a_1, a_2 \in \mathcal{A}$, we write $a_1 \sqsubseteq a_2$ iff either (a) $a_1 = \epsilon$, or (b) $a_1 = f(a_1^1, a_2^1, \ldots, a_k^1)$ and $a_2 = f(a_1^2, a_2^2, \ldots, a_k^2)$ and $a_1^1 \sqsubseteq a_1^2$, $a_2^1 \sqsubseteq a_2^2$, $\ldots$, $a_k^1 \sqsubseteq a_k^2$.

Intuitively, $a_1 \sqsubseteq a_2$ when a_1 can be obtained from a_2 by substituting ϵ for zero or more branches in a_2. We say that a_1 is a ***subtree*** of a_2.

By definition, $\sqsubseteq$ is a partial order: i.e., reflexive, transitive and antisymmetric. (Note that, as a special case, $a_1 = a_2$ implies $a_1 \sqsubseteq a_2$.) We write $a_1 \sqsubset a_2$ iff $a_1 \sqsubseteq a_2$ but $a_1 \neq a_2$. Since trees are finite, there is no infinite descending chains of the form $a \sqsupset a_1 \sqsupset a_2 \sqsupset \ldots$.

4.2.3 Tree transformation

In the following, we define operators for specifying transformations of tree-structured profiles.

Definition 5 (Fragment: $\mathbb{N}^* \rightharpoonup \mathcal{A}$). *A fragment σ is a finite subset of $\mathbb{N}^* \times \mathcal{A}$ that satisfies two further properties. First, σ is a function. (Thus we write $\sigma(p)$ for the tree associated with position p, and $dom(\sigma)$ for the domain of σ.) Second, the positions in $dom(\sigma)$ are pairwise incomparable.*

The set of all possible fragments is denoted by $\mathcal{V}$. A fragment σ is ***compatible*** to a tree a if for every $p \in dom(\sigma)$, $p \in \mathcal{P}os(a)$ and $a \downarrow p = \epsilon$.

Given a fragment σ and a position p, the projection $\sigma|_p$ is defined to be the fragment $\{(q, a) \mid (p \cdot q, a) \in \sigma\}$. Note that as special cases p can be either ϵ or a sequence of length one.

Definition 6 (Append Operator $\oplus : \mathcal{A} \times \mathcal{V} \to \mathcal{A}$). *Let $a \in \mathcal{A}$, and $\sigma \in \mathcal{V}$ be a compatible fragment. Then $a \oplus \sigma$ is the tree defined inductively as follows: (a) $\epsilon \oplus \sigma = \sigma(\epsilon)$, and (b) $f(a_1, \ldots, a_k) \oplus \sigma = f(a_1 \oplus (\sigma|_1), \ldots, a_k \oplus (\sigma|_k))$.*

Intuitively, $a \oplus \sigma$ is the tree obtained from a by the following substitutions: for every position p in the domain of σ, the tree $\sigma(p)$ is inserted in position p of a. Note that for every $a \in \mathcal{A}$, and every fragment $\sigma \in \mathcal{V}$ that is compatible to a, $a \sqsubseteq a \oplus \sigma$ holds.

Definition 7 ($\setminus : \mathcal{A} \times \mathcal{A} \to \mathcal{V}$). *Suppose $a_1, a_2 \in \mathcal{A}$ and $a_2 \sqsubseteq a_1$. Let $P = \{p \in \mathcal{P}os(a_1) \cap \mathcal{P}os(a_2) \mid a_1 \downarrow p \neq a_2 \downarrow p\}$. We write $a_1 \setminus a_2$ to denote the fragment $\{(p, a_1 \downarrow p) \mid p \in P\}$.*

In the above definition, P is the set of positions that are filled in a_1 but empty in a_2. The fragment $a_1 \setminus a_2$ is essentially the "delta" between a_1 and a_2. By definition, for every $a_1, a_2 \in \mathcal{A}$, if $a_2 \sqsubseteq a_1$, then $a_1 = a_2 \oplus (a_1 \setminus a_2)$. Note that $a_1 \setminus a_2$ is defined only if $a_1 \sqsubseteq a_2$.

5. TRANSFORMATION FUNCTIONS

In §3, we studied the privacy enhancing and utility preserving effects of abstract transformations. Now armed with the algebraic tools of §4.2, the goal of this section is to apply the assessment framework of §3 to study concrete sanitizing operations for semi-structured data. To do so, we define a pattern for transformation functions.

Conceptually, given a tree, a transformation typically makes local changes in some branches of the tree while the rest of the tree remains intact. This means, there needs to be a way to identify the branches that are to be manipulated as well as the parts that will not change. For that purpose, we define ***reduction*** as follows:

Definition 8 (Reduction Function). *A deterministic function $r : \mathcal{A} \to \mathcal{A}$ is a* **reduction** *iff $\forall a \in \mathcal{A} \,.\, r(a) \sqsubseteq a$.*

Intuitively, $r(a)$ is a subtree of a that will remain unchanged, while the fragment $a \setminus r(a)$ captures parts of a that will undergo further sanitization. Accordingly, we have:

$$\forall p \in dom(a \setminus r(a)) \,.\, r(a) \downarrow p = \epsilon$$
$$\forall p \in \mathcal{P}os(r(a)) \setminus dom(a \setminus r(a)) \,.\, r(a) \downarrow p = a \downarrow p$$

Since $r(a) \sqsubseteq a$, we also know that $a = r(a) \oplus (a \setminus r(a))$.

After identifying the fragment that is to be transformed, we need to first sanitize the fragment and then graft it back to the reduced tree. This implies the need for a ***fragment transformation function***, which transforms a given fragment to another fragment. Formally, a fragment transformation $t_f : \mathcal{V} \to \mathcal{V}$ is a function such that $dom(t_f(\sigma)) = dom(\sigma)$ for all $\sigma \in \mathcal{V}$. We now define a template for tree transformations using our algebra of trees.

Definition 9 (Transformation Function). *$t : \mathcal{A} \to \mathcal{A}$ is a transformation function iff there exists a reduction r and a fragment transformation function t_f such that for every $a \in \mathcal{A}$, $t(a) = r(a) \oplus t_f(a \setminus r(a))$.*

Fragment $a \setminus r(a)$ is transformed by t_f and then grafted back onto $r(a)$, the unchanged subtree of a. Now given A as a profile, both $r(A)$ and $A \setminus r(A)$ are random variables.

The above definition could cover variety of transformations because it imposes no constraint on t or t_f. In the following sections, we will provide concrete transformations based on the above definition.

Core of Correlation. Correlation between the sensitive information and the accessible information usually could be attributed to some parts of the accessible information such that if we remove those parts, there will be no correlation between the sensitive and accessible information. We call such parts to be the ***core of correlation*** between the two pieces of information. More precisely, given that there is some correlation between S and A (i.e., $S_A > 0$), a fragment $A \setminus r(A)$ of A is called the core of correlation between S and A if the following holds:

$$S_{r(A)} = I(S; r(A)) = 0 \tag{10}$$

This notion of a core provides guidance on the engineering of sanitizing transformations. To break the correlation between a profile A and sensitive information S, one can identify the core of correlation by specifying an appropriate reduction function r. Once the core of correlation $A \setminus r(A)$ is identified, one needs to only focus on transforming the core (via a fragment transformation) since the rest ($r(A)$) does not contain any information about S. Transformation $t(A) = r(A) \oplus t_f(A \setminus r(A))$ is called a ***core transformation*** iff $A \setminus r(A)$ is the core of correlation between A and S.

Privacy-Enhancing Transformation. Identifying the core of correlation between S and A, and then engineering a core transformation is a significant step towards an effective protection of S. Nevertheless, there is no guarantee that a core transformation t is definitely privacy enhancing because how the core $A \setminus r(A)$ is transformed is of importance too.

A transformation t is ***privacy-enhancing*** w.r.t S and A iff $S_{t,A} > 0$ (i.e., $I(S; A) > I(S; t(A))$).

Proposition 5. *Given profile A, sensitive information S, and reduction r, transformation $t(A) = r(A) \oplus t_f(A \setminus r(A))$ is privacy-enhancing if $I(S; A \setminus r(A) \,|\, t(A)) > 0$.*

Intuitively, if we can identify a reduction r such that the fragment $A \setminus r(A)$ has some mutual information with S that is not contained in $t(A)$ (i.e., $I(S; A \setminus r(A) \mid t(A)) > 0$), then $t(A)$ will have strictly less correlation with S. The reason we emphasize that such mutual information must not be contained in $t(A)$ is that otherwise $t(A)$ would still carry the information of S that is contained in the transformed fragment. As a result, there would be no guarantee that $I(S; t(A)) < I(S; A)$. A proof of the above result is given in [2, Proposition 4.6.3].

Implementation of concrete transformations. Having a pattern for transformation functions, defined in Definition 9,

we can now show by examples how semi-structured analogues of classical transformations, previously employed for sanitizing structured data, can be instantiated.

EXAMPLE 1 (SUPPRESSION, AND GENERALIZATION). *Suppression and generalization have been widely used in the literature as simple techniques for sanitizing data tables. The former partially removes parts of the information, whereas the latter replace parts of the information with less specific information. Let $t_l : \mathcal{A} \rightharpoonup \mathcal{A}$, which we call a* **deterministic local transformation function***, be a partial function mapping trees to trees. Assume a deterministic fragment transformation function t_f that is induced by t_l as follows:*

$$t_f(\sigma)(p) = \begin{cases} t_l(\sigma(p)) & \text{if } \sigma(p) \in dom(t_l) \\ \sigma(p) & \text{otherwise} \end{cases}$$

Intuitively, t_f alters a fragment σ by going through each position p in the domain of σ; if t_f finds a value $\sigma(p)$ within the domain of t_l, then t_f transforms that value to $t_l(\sigma(p))$; otherwise the value is left as is for position p. Now if in the above definition of t_f, we choose t_l to map every tree to ϵ, then transformation function t, resulted by Definition 9, will be a **suppression**. *However, if t_l maps trees in its domain to their generalized form, the resulting transformation t will be a* **generalization**. *This latter local transformation function t_l is usually called a* value generalization hierarchy.

EXAMPLE 2 (PERMUTATION). *In the literature, a probabilistic sanitizing technique known as swapping switch values in data tables. For example, given a data table, the value of an attribute in a row is substituted with the value of the same attribute from another row. Therefore, only the positions where values are located change. Such transformations are called permutation in this work. In the context of semi-structured profiles, permutation means randomly switching the places of two or more branches in a tree. To define permutation, we just need to define the appropriate probabilistic fragment transformation. The transformed profile will be constructed by grafting the permuted fragment back to the reduced profile. Given a random variable R (serving as a source of randomness), $t_{R,f} : \mathcal{V} \to \mathcal{V}$ is a probabilistic fragment transformation if for every $\sigma \in \mathcal{V}$, $t_{R,f}(\sigma) = \sigma \circ \pi_R$, where $\pi_R : dom(\sigma) \to dom(\sigma)$ is a bijection generated from the random source R. In other words, $t_{R,f}(\sigma) = \{(\pi_R^{-1}(p), a) \mid (p, a) \in \sigma\}$. Recalling Definition 9, where $t(a) = r(a) \oplus t_f(a \setminus r(a))$, probabilistic transformation t is a* **permutation function** *iff t_f is a fragment permutation.*

EXAMPLE 3 (NOISE INTRODUCTION). *In the context of semi-structured data, there are two types of noise introduction. One is to replace branches in a tree-shaped profile with randomly noisified branches. Since, this would be very similar to generalization, we will not elaborate more on that. Another technique is to inject noise into the tree in the form of new branches. For instance, in the birthday example, we could confuse the adversary by adding birthday greetings on random days to the user's wall. In this work, this type of noise introduction is called* **noise addition**. *Let R be a source of randomness that allows us to randomly select a reduced profile $a' = r_R(a)$, where a is the original profile, and r_R is a probabilistic reduction function with random variable R ($\sigma = a \setminus a'$ is the extracted fragment). Next, we sample from another random variable N to obtain a number of noisy branches. Now we need a fragment transformation called* **fragment expansion***, denoted by $\cdot \lhd \cdot : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$, that uses the noisy branches to mutate the σ, before the latter is grafted back onto the reduced profile a'. Suppose $\sigma = \{(p_1, a_1), \ldots, (p_n, a_n)\}$ and $\sigma^\star = \{(p_1^\star, a_1^\star), \ldots, (p_n^\star, a_n^\star)\}$ are fragments from $\mathcal{V}$ with equally-sized domains, such that $p_i^\star \in \mathcal{P}os(a_i^\star)$ and $a_i^\star \downarrow p_i^\star = \epsilon$ for $1 \le i \le n$. Then $\sigma^\star \lhd \sigma$, called* **the expansion of σ with $\sigma^\star$**, *is the fragment $\{(p_1, a_1^\star \oplus \{(p_1^\star, a_1)\}), \ldots, (p_n, a_n^\star \oplus \{(p_n^\star, a_n)\})\}$. A fragment transformation $t_{N,f} : \mathcal{V} \to \mathcal{V}$ is a* **fragment expansion** *iff $t_{N,f}(\sigma)$ returns $\sigma^\star \lhd \sigma$, where $\sigma^\star$ is generated probabilistically using random variable N as a source of randomness. Now given random variables R and N, the probabilistic function $t_{R,N}$ is a* **noise addition** *if there exists probabilistic reduction function r_R and fragment expansion $t_{N,f}$ such that, for every $a \in \mathcal{A}$, $t_{R,N}(a) = a' \oplus t_{N,f}(a \setminus a')$, where $a' = r_R(a)$. Note that this definition follows the pattern of Definition 9.*

Note that the sources of randomness in permutation and noise addition must be selected carefully to not violate the conditional independence required in the universal inference rules (Table 1). This was thoroughly discussed in §3.3.2.

Discussion. In this section, we defined a transformation function to be 1) identifying a fragment of a user profile, 2) transforming the fragment, and 3) grafting back the transformed fragment onto the rest of the profile. We also showed that practical sanitizing techniques (e.g., noise addition) could be realized using this definition. Now given that our privacy enhancement and utility degradation measures are additive as illustrated in §3.6, we can design multiple transformation functions based on Definition 9. If every one of such transformations meet the requirements of the proposed universal inference rule in Table 1, we could completely predict the behaviour of the composition of those transformation functions, and calculate the overall privacy enhancement and utility degradation. Moreover, we further employed the proposed constructs of §3, and showed that identifying the core of correlation could potentially result in an effective transformation. The accurate notion of privacy enhancing transformation was also proposed to describe the condition that, if met, guarantees privacy enhancement. On top of that, once a transformation is designed, our measures of unnecessary information leakage and transformation, proposed in §3.4.3 and §3.4.4, will help to make sure if the transformation has been properly designed. If so, as proved in §3.5, the maximum amount of information leakage and utility degradation will be bounded by $I(S;U)$.

6. EVALUATION

In this section, we evaluate two aspects of our framework. First, §6.1 presents a case study illustrating the usefulness of our framework for evaluating inference control mechanisms. Second, §6.2 uses our framework to characterize the properties of the four examples of sanitizing transformations.

6.1 User Profile Transformation – Example

6.1.1 Components

User profiles. In our example social network, a user has 2 friends c and d. Attribute B in the user's profile shows the name of the user's partner, who is either friend c or friend d, i.e., $D_B = \{c, d\}$. There is also a photo album in the user's profile with two photos: 1) a wedding photo, and 2) a

$P(S\|B)$	S	
B	c	d
c	1	0
d	0	1

Table 4: Conditional distribution of S given B.

$P(T\|B)$	T			
B	$c_w d_n$	$c_n d_w$	$c_n d_n$	$c_w d_w$
c	0.6	0.1	0.1	0.2
d	0.1	0.6	0.1	0.2

Table 5: Conditional distribution of T given B.

nature photo. Every friend of the user is tagged in either the wedding or the nature photo. T represents the user's photo album such that $D_T = \{c_w d_n, c_n d_w, c_n d_n, c_w, d_w\}$ where, for example, $c_w d_n$ means friend c is tagged in the wedding photo and friend d is tagged in the nature photo. Therefore, the user profile A is simply modelled using the joint distribution of the two variables B and T, i.e, $A = B, T$.

Sensitive information. The sensitive information S that is to be inferred by an adversary from the user profile is the name of the user's partner. In other words, S and B are equal in our example (and hence $D_S = D_B$).

Sample third-party extension. Consider a third-party extension that finds the total number of friends who are tagged in a user's photo album. This would be the utility U, that the extension provides for its users.

Correlation between information. In our example, S, which is to be protected, is a deterministic function of A because A contains B which is basically equal to S. Similarly, U is also a deterministic function of T, and as a result, A. It is clear that B must not be included in the sanitized version of A, otherwise S could be inferred with complete certainty. The trickier correlation is the one between S and T. If a friend is tagged in a user's wedding photo, he/she is likely to be the user's partner. As a result, T could partially reveal S. Note that T is required by the extension for computing U.

Tables 4 and 5 describe the correlation between the random variables in our sample profile. Note that $A = B, T$ and $H(S \mid B, T) = H(S \mid B) = 0$.

6.1.2 Transformation

We propose two transformations for hiding S while preserving U.

Suppression. In our sample social network, the most confident way to infer S from a profile A is to use B because $H(S \mid B) = 0$. As a result, $H(S \mid A)$ is 0, showing the necessity of transforming A to protect S. Since there is no correlation between U and B, we could easily remove B from A to break the correlation between A and S. To do so, we employ a suppression t_1 such that $A \setminus t_1(A)$ always points to B, which identifies the user's partner. This means $t_1(A)$ does not contain B. As a result, $t_1(A)$ would be equal to T. The universal inference rule guarantees that $t_1(A)$ contains no more information about S than A does. To show that t_1 is really effective in protecting S, we compute $S_{t_1,A}$. Let's assume $P(S = 1) = P(S = 2) = 0.5$.

$$\begin{aligned} S_A &= H(S) - H(S \mid A) = H(S) - H(S \mid B, T) = 1 \\ S_{t_1(A)} &= I(S; t_1(A)) = I(S; T) = 0.283 \\ S_{t_1,A} &= 1 - 0.283 = 0.717 \end{aligned} \tag{11}$$

$P(T' \mid T)$	T'			
T	$c_w d_n$	$c_n d_w$	$c_n d_n$	$c_w d_w$
$c_w d_n$	0.5	0.5	0	0
$c_n d_w$	0.5	0.5	0	0
$c_n d_n$	0	0	1	0
$c_w d_w$	0	0	0	1

Table 6: Conditional distribution of T' given T
.

$P(S \mid T)$	S	
T	c	d
$c_w d_n$	0.86	0.14
$c_n d_w$	0.14	0.86
$c_n d_n$	0.5	0.5
$c_w d_w$	0.5	0.5

Table 7: Conditional distribution of S given T.

$P(S \mid T')$	S	
T'	c	d
$c_w d_n$	0.5	0.5
$c_n d_w$	0.5	0.5
$c_n d_n$	0.5	0.5
$c_w d_w$	0.5	0.5

Table 8: Conditional distribution of S given T'.

$P(S \mid T)$, which is needed in the above calculations, is given in Table 7. Since the evaluated value of $S_{t_1,A}$ is close to its maximum value (which is 1 in this example), the privacy enhancement caused by the suppression t_1 is significant. We skip computing the utility degradation as t_1 does not make any change in T, implying that $U_{t_1,A} = 0$.

Permutation. Although $S_{t_1,A} = 0.717$ shows a considerable amount of privacy enhancement, there is still a chance that S could be inferred from $t_1(A)$ due to the correlation between S and T. To address this issue, we employ a permutation t_2 to randomly permute the photo tags such that the adversary cannot be certain if, for example, friend c was originally tagged in the wedding photo or the nature photo. Given that the input to t_2 is the output of the suppression $t_1(A)$ which was equal to T, assume T' is the output of $t_2(t_1(A))$. Distribution $P(T' \mid T)$ in Table 6 shows how the permutation t_2 works. If a friend is tagged in a photo, after permutation, with probability 0.5, he/she will be tagged in the other photo, and with probability 0.5 he/she will be tagged in the same photo as before. The privacy enhancement caused by the permutation t_2 is assessed as follows:

$$\begin{aligned} S_{t_1(A)} &= 0.283 \text{ according to (11)} \\ S_{t_2(t_1(A))} &= H(S) - H(S \mid t_2(t_1(A))) = 1 - H(S \mid T') = 0 \\ S_{t_2,t_1(A)} &= 0.283 - 0 = 0.283 \end{aligned}$$

Table 8 shows $P(S \mid T')$ that is needed in the above calculation.

Since our measure of privacy enhancement is additive, the total privacy enhancement caused by $t = t_2 \circ t_1$ over A is computed as follows:

$$S_{t,A} = S_{t_1,A} + S_{t_2,t_1(A)} = 0.717 + 0.283 = 1$$

As a result, t completely hides S because it caused the maximum amount of privacy enhancement, i.e., $S_{t,A} = I(S; A) = 1$. In designing the second transformation, we were careful-

enough not to change the total number of tags. Hence, $U_{t,A}$ would be 0 showing that t causes no utility degradation.

6.2 Transformation Analysis

We surveyed four examples (classes) of sanitizing transformations for semi-structured data: suppression, generalization, permutation, and noise addition. When would one class be preferred over another? In the following, we use our analytical framework to highlight the characteristics of each class. Transformations can be compared in terms of (a) their ability to preserve utility, (b) their effectiveness on protecting privacy, and (c) their efficiency. We will focus on (a) and (b), as (c) is outside of the scope of this work.

Common to the four transformation classes is a reduction function r that factorizes the profile a into two parts: one that is released as is ($r(a)$), and a fragment $a \setminus r(a)$ that will undergo further sanitization via a fragment transformation t_f. The designer of the sanitizing transformation now has to decide if he is to release only the unchanged part of the profile (suppression), or to release it along with a generalized, permuted, or noisified version of the fragment $a \setminus r(a)$. To make the comparison fair, we assume that the four choices are based on the same reduction r, and that $a \setminus r(a)$ captures the core of correlation (§5).

Core suppression – least information leakage. As $a \setminus r(a)$ is the core of correlation, completely removing it by suppression would entirely eliminate any correlation with the sensitive information S. In the other three transformations, $t(a) = r(a) \oplus t_f(a \setminus r(a))$ is released, and thus some correlation with S may remain in $t(a)$. This implies that a core suppression t would contain the least sensitive information compared to the other three transformations. Therefore, core suppression is always the best choice for enhancing privacy, but the worst choice for preserving utility.

Permutation and aggregate utilities. Imagine we divide a profile into two parts: (a) values, and (b) structure that captures the association among values. In our algebraic structure, constant symbols are the values, and function symbols with one or more input arguments contribute to the structure of profile. For instance in Table 3, `1/2/2013` and `hello` are two values, and the function symbol `entry` connect them to indicate that `1/2/2013` is the timestamp of a wall entry with the message `hello`.

Suppression could potentially remove both structure and values. Generalization only changes values but leaves the structure unchanged. Noise addition changes both values and structure. In contrast, permutation is the only transformation that manipulates only the structure of profiles, and leaves the values as they are. A corollary to this observation is that, if the utility of a profile resides only on the values, permutation may be the best choice as far as utility preservation is concerned. This usually happens when utility U is in the form of some aggregate of values: e.g., counting the total number of entries. The output of such queries remains the same after permutation.

7. RELATED WORK

Privacy in general has always been a concern of security experts. In particular, lots of efforts have been put to protect the identity of people in datasets where information belong to individuals. To evaluate such efforts that are usually called anonymization, measures have been defined to assess to what extent the identity of individuals is protected when a dataset is released. Measures such as k-anonymity [21], l-diversity [18], t-closeness [15], etc., are the most well-known metrics. In a more recent work by Askari *et al.* [6], an information theoretic framework is proposed to translate all these anonymization techniques to a unified system.

Although the above anonymization techniques are well-explored in privacy preserving data mining applications, they have not been employed in the context of SNSs. Unlike data mining applications, in SNSs the identity of users are usually known to other users as well as third-party extensions. In fact, known identities is a part of the culture of social networks. Third-party developers expect to know their users' identity. SNSs utilized permission-based authorization mechanisms to protect their users' information. Nonetheless, it has been proved by several works that the privacy concerns still exist in SNSs. For example, Bilge *et al.* [7] discuss two adversarial techniques to establish friendship relations with users and access their information. Similarly, Boshmaf *et al.* [9] showed that their socialbots could impersonate a human being, and then access users' information by establishing friendship relation with them.

Unlike inference problem in databases, reported many years ago in works such as [20] and [23], inference attacks in SNSs is a hot research area. The reason lies behind the wide popularity of social networks. Zheleva and Getoor [25] conducted an experiment to demonstrate the feasibility of inference attacks in social network data sets. Moreover, Xu *et al.* [24] proposed a work very similar to [25] in which social friendship information is used to infer an individual's gender.

The role of third-party extensions in launching inference attacks has been completely overlooked in the literature. Previously published empirical studies [25, 24, 13] assume the adversary has access to the whole social network data set. This assumption is simply not realistic in the case when inference attacks are launched via third-party applications. Ahmadinejad *et al.* [4, 5] were the first who evaluated the threat of inference attacks by third-party extensions to SNSs. Such efforts can bridge the gap between users's expectation from authorization mechanisms in SNSs and what really happens behind the scene. Although those efforts are very helpful in warning users to carefully share information with extensions, there is still a need for protection mechanisms particularly designed for controlling inference made by third-party extensions, which is the focus of this work.

A different perspective in dealing with privacy issues in social networks is distrusting all involved parties including the SNS providers. To operationalize this idea, [16, 17, 11] proposed cryptographic techniques that allow SNS providers to only access an encrypted form of user data. Authorized users will have the required keys for decrypting the data.

8. CONCLUSION AND FUTURE WORK

We articulated the need of view-based protection for preventing inference attacks by third-party applications in social computing platforms, and proposed a theoretical framework for assessing if sanitizing transformations of semi-structured data meet quantitative privacy and utility goals. We also identified the conditions under which transformations are safely composable. We modelled classical sanitizing transformations as operations over term algebras of trees, and demonstrated how their privacy enhancing and utility preserving effects is evaluated in our framework. Our work therefore offers a means to formally verify the privacy

and utility properties of transformations for controlling inference by applications that process semi-structured data.

This work is part of a larger research program to develop inference control mechanisms for programmable environments. Our current work involves the development of concrete protection technologies that interoperate with existing tree manipulation and querying standards (e.g., XML), as well as query optimization technologies that avoid materializing views before evaluating the queries. Future work also include the strengthening of privacy and utility goals.

Acknowledgments

This work is supported in part by an NSERC Discovery Grant (RGPIN-2014-06611) and a Canada Research Chair (950-229712).

9. REFERENCES

[1] Aggarwal, C. C., and Yu, P. S. *Privacy-Preserving Data Mining: Models and Algorithms*. Springer, 2008.

[2] Ahmadinejad, S. H. *A View-Based Protection Model to Prevent Inference Attacks by Third-Party Extensions to Social Computing Platforms*. PhD dissertation, University of Calgary, Jan. 2016. http://hdl.handle.net/11023/2755.

[3] Ahmadinejad, S. H., Anwar, M., and Fong, P. Inference attacks by third-party extensions to social network systems. In *Proceedings of IEEE 9th International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)* (2011), pp. 282 –287.

[4] Ahmadinejad, S. H., and Fong, P. W. On the feasibility of inference attacks by third-party extensions to social network systems. In *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security* (Hangzhou, China, 2013), ASIACCS'13, pp. 161–166.

[5] Ahmadinejad, S. H., and Fong, P. W. Unintended disclosure of information: Inference attacks by third-party extensions to social network systems. *Computers & Security 44* (2014), 75–91.

[6] Askari, M., Safavi-Naini, R., and Barker, K. An information theoretic privacy and utility measure for data sanitization mechanisms. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy* (San Antonio, Texas, USA, 2012), CODASPY'12, pp. 283–294.

[7] Bilge, L., Strufe, T., Balzarotti, D., and Kirda, E. All your contacts are belong to us: automated identity theft attacks on social networks. In *Proceedings of the 18th International Conference on World Wide Web* (Madrid, Spain, 2009), WWW'09, pp. 551–560.

[8] Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., Siméon, J., and Stefanescu, M. XQuery 1.0: An XML query language. *W3C Recommendation* (2007).

[9] Boshmaf, Y., Muslukhov, I., Beznosov, K., and Ripeanu, M. The socialbot network: when bots socialize for fame and money. In *Proceedings of the 27th Annual Computer Security Applications Conference* (Orlando, FL USA, 2011), ACSAC'11, pp. 93–102.

[10] Cover, T. M., and Thomas, J. A. *Elements of Information Theory*. Wiley-Interscience, 2006.

[11] De Cristofaro, E., Soriente, C., Tsudik, G., and Williams, A. Hummingbird: privacy at the time of Twitter. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy* (San Francisco, CA, USA, 2012), SP'12, pp. 285–299.

[12] Gries, D. *The Science of Programming*. Springer-Verlag, 1981.

[13] He, J., Chu, W. W., and Liu, Z. V. Inferring privacy information from social networks. In *Proceedings of the 4th IEEE International Conference on Intelligence and Security Informatics* (San Diego, CA, USA, 2006), ISI'06, Springer-Verlag, pp. 154–165.

[14] Kay, M. XML Path language (XPath) version 2.0. *W3C Recommendation 23* (2007).

[15] Li, N., Li, T., and Venkatasubramanian, S. t-Closeness: privacy beyond k-anonymity and l-diversity. In *Proceedings of the 23rd IEEE International Conference on Data Engineering* (Istanbul, Turkey, 2007), ICDE'07, pp. 106–115.

[16] Lucas, M. M., and Borisov, N. FlyByNight: mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society* (Alexandria, VA, USA, 2008), WPES'08, pp. 1–8.

[17] Luo, W., Xie, Q., and Hengartner, U. FaceCloak: an architecture for user privacy on social networking sites. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering* (Vancouver, Canada, 2009), vol. 3 of *CSE'09*, pp. 26–33.

[18] Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. l-diversity: privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data 1*, 1 (2007), 1–52.

[19] Pearl, J. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, NY, USA, 2000.

[20] Robling Denning, D. E. Inference controls. In *Cryptography and data security*. Addison-Wesley Longman Pub. Co., 1982, ch. 6, pp. 331–390.

[21] Samarati, P. Protecting respondents identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering 13*, 6 (2001), 1010–1027.

[22] Smith, G. Quantifying information flow using min-entropy. In *Proceedings of the 8th International Conference on Quantitative Evaluation of Systems* (2011), QEST'11, pp. 159–167.

[23] Thuraisingham, B. M. A perspective of the inference problem. In *Database and applications security*. 2005, ch. 12, pp. 217–228.

[24] Xu, W., Zhou, X., and Li, L. Inferring privacy information via social relations. In *Proceedings of the 24th IEEE International Conference on Data Engineering Workshop* (Cancun, Mexico, 2008), ICDEW'08, pp. 525–530.

[25] Zheleva, E., and Getoor, L. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of the 18th International Conference on World Wide Web* (Madrid, Spain, 2009), WWW'09, pp. 531–540.