This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

PPTP: Privacy-Preserving Traffic Padding in Web-Based Applications

Wen Ming Liu, Lingyu Wang, Pengsu Cheng, Kui Ren, Shunzhi Zhu and Mourad Debbabi

***Abstract*— Web-based applications are gaining popularity as they require less client-side resources, and are easier to deliver and maintain. On the other hand, Web applications also pose new security and privacy challenges. In particular, recent research revealed that many high profile Web applications might cause sensitive user inputs to be leaked from encrypted traffic due to side-channel attacks exploiting unique patterns in packet sizes and timing. Moreover, existing solutions, such as random padding and packet-size rounding, were shown to incur prohibitive overhead while still failing to guarantee sufficient privacy protection. In this paper, we first observe an interesting similarity between this privacy-preserving traffic padding (PPTP) issue and another well studied problem, privacy-preserving data publishing (PPDP). Based on such a similarity, we present a formal PPTP model encompassing the privacy requirements, padding costs, and padding methods. We then formulate PPTP problems under different application scenarios, analyze their complexity, and design efficient heuristic algorithms. Finally, we confirm the effectiveness and efficiency of our algorithms by comparing them to existing solutions through experiments using real-world Web applications.**

***Index Terms*— Traffic Padding, Web Application, Side-Channel Leak, PPTP, l-Diversity, k-Indistinguishability**

I. Introduction

Web-based applications are becoming increasingly popular. In contrast to their desktop counterparts, Web applications demand less client-side resources and are easier to deliver and maintain through using the Web browser as a thin client. On the other hand, Web applications also present new security and privacy challenges, partly because the untrusted Internet has essentially become an integral component of such applications for carrying the continuous interaction between users and servers. Recent study showed that the encrypted traffic of many popular Web applications may actually disclose highly sensitive data, and consequently lead to serious breaches of user privacy [14]. Specifically, by searching for unique patterns exhibited in packets' sizes and/or timing, an eavesdropper can potentially identify an application's internal state transitions and the corresponding users' inputs. Moreover, such side-channel attacks are shown to be pervasive and fundamental to most Web applications due to many intrinsic characteristics of such applications, such as low entropy inputs, diverse resource objects, and stateful communications.

Taking one popular real-world search engine as an example, Table I shows the sizes and directions of packets observed between users and the search engine. Observe that due to the user-friendly *auto-suggestion* feature [1], with each keystroke, the browser sends a b-byte packet to the server; the server then replies with two packets of 60 bytes and s bytes, respectively; finally, the browser sends a 60-byte packet to the server. In addition, in the same input string, the b value of the first keystroke is about 50 bytes larger than that of the second one while each subsequent keystroke increases the b value by one byte from the third keystroke, and the s value depends both on the current keystroke and on all the preceding ones. Clearly, due to the fixed pattern in packet sizes (first, second, and last), the packets corresponding to each input string can be identified from observed traffic, even though the traffic has been encrypted.

User Input	Observed Directional Packet Sizes			
bee	641 →,	← 60,	← 544,	60 →,
	585 →,	← 60,	← 555,	60 →,
	586 →,	← 60,	← 547,	60 →
cab	641 →,	← 60,	← 554,	60 →,
	585 →,	← 60,	← 560,	60 →,
	586 →,	← 60,	← 558,	60 →
	(b bytes)		(s bytes)	

TABLE I
User Inputs and Corresponding Packet Sizes

Similar traffic patterns have also been observed in different categories of Web applications [14]. Therefore, we assume a worst case scenario in which an eavesdropper can pinpoint traffic related to a Web application (such as using de-anonymizing techniques [37]) and locate packets for user inputs using the above technique. We use search engines as examples in this paper due to their distinct and representative patterns. In reality, the s value can be larger and more disparate as discussed in Section VI.

Moreover, the size of the third packet provides a good indicator of the input itself (which again can be found in many Web applications [14]). Specifically, Table II shows the s value for character (a, b, c and d) entered as the first (second column) and second (3-6 columns) keystroke for a different search engine. Observe that the s value for each character entered

W. M. Liu, L. Wang, M. Debbabi are with Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC H3G 1M8, Canada. E-mail:{l_wenmin,wang,debbabi}@ciise.concordia.ca.

P. Cheng is with the Gameloft Inc., Montreal, QC H2S 3L4, Canada. E-mail:chengpengsu@gmail.com.

K. Ren is with Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, New York 14260, USA. E-mail:kuiren@buffalo.edu.

S. Zhu is with Department of Computer Science and Technology, Xiamen University of Technology, Xiamen 361024, PR China. E-mail:szzhu@xmut.edu.cn.

[1] Note the discussion of auto-suggestion feature here does not imply the side channel attack to be only applicable to Web applications with this feature; the feature is just an example of the stateful communications inherent to most Web applications, which is one of the root causes of the side channel attack [14].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

as second keystroke is different from that it is entered as the first, since the packet size now depends on both the current keystroke and the preceding one. Clearly, every input string can be uniquely identified by combining observations of packet sizes about the two consecutive keystrokes (for simplicity, we only consider $a-d$ combinations here, whereas in reality it may take more than two keystrokes to uniquely identify an input string).

		Second Keystroke			
First Keystroke		**a**	**b**	**c**	**d**
a	509	487	493	501	497
b	504	516	488	482	481
c	502	501	488	473	477
d	516	543	478	509	499

TABLE II

s VALUE FOR EACH CHARACTER ENTERED AS THE FIRST (SECOND COLUMN) AND SECOND (3-6 COLUMNS) KEYSTROKE

A natural solution for preventing such a side channel attack is to pad packets such that each packet size will no longer map to a unique input (One extreme case is to pad all packets to the identical size, namely, *maximizing*). In this example, we should pad s-byte such that each packet size maps to at least $k=2$ different inputs, namely, 2-*indistinguishability*.

However, such a solution does not come free, since padding packets will result in additional communication and processing overhead. In fact, it has been shown that a straightforward solution, such as random padding (appending a random-length padding within a given interval to a packet) and rounding (rounding packet sizes to the nearest intervals), may incur a prohibitive overhead. Thus, we face two seemingly conflicting goals. First, the difference in packet sizes needs to be sufficiently reduced to prevent eavesdroppers from distinguishing between different users inputs based on corresponding packet sizes. Second, the overhead for achieving such privacy protection should be minimized. Finally, a tradeoff naturally exists between these two objectives.

We now consider a different way for padding the packets as shown in Table III. The first and last columns respectively show the s value and corresponding character with its prefix (e.g., $(c)d$ means the character d is entered as the second keystroke after its prefix c is entered for the same input string). The middle two columns give two options for padding packets (although not shown here, there certainly exist many other options). Specifically, each option first divides the six keystrokes into three (or two) *padding groups*, as illustrated by the (absence of) horizontal lines. Packets within the same padding group are then padded in such a way that the corresponding s values become identical to the maximum value in that group, and thus the characters inside the group will no longer be distinguishable from each other by the s values. The objective now is to find a padding option that can provide sufficient privacy protection groups, while at the same time and meanwhile minimize the padding cost. Note that gathering such packet information is practical for most Web applications, as we will discuss later in Section VI-A. Interestingly, as we will show in Section II-B, this *privacy-preserving traffic padding (PPTP)* problem is naturally associated with another well studied problem, namely, *privacy-preserving data publishing (PPDP)* [20]. Such a connection between the two issues implies we may borrow many existing efforts in the PPDP domain to address the PPTP issue.

s **Value**	**Padding**		**(Prefix)Char**
	Option 1	**Option 2**	
473	477	478	$(c)c$
477	477	478	$(c)d$
478	499	478	$(d)b$
499	499	509	$(d)d$
501	509	509	$(c)a$
509	509	509	$(d)c$
Quasi-ID	**Generalization**		**Sensitive Value**

TABLE III

MAPPING PPTP TO PPDP

In this paper, we first present a model of the PPTP issue based on the mapping to PPDP, which formally characterizes the interaction between users and Web applications, the observation made by eavesdroppers, the privacy requirement, and the overhead of padding. Based on the model, we then formulate several PPTP problems under different assumptions, and discuss the complexity. We show that minimizing padding cost under a given privacy requirement is generally intractable. Next, we design several heuristic algorithms for solving the PPTP problems in polynomial time with acceptable overhead. Finally, we demonstrate the effectiveness and efficiency of our algorithms by both analytical and experimental evaluations.

The contribution of this paper is threefold. First, the identified similarity between PPTP and PPDP establishes a bridge between the two research areas, which will not only allow for reusing many existing models and methods in the well investigated PPDP domain, but serve to attract more interest to the important PPTP issue. Second, to the best of our knowledge, our formal model is among the first efforts on formally addressing the PPTP issue (a detailed review of related work will be given in Section VII). Third, the proposed algorithms may provide direct and practical solutions to real world PPTP applications, as evidenced by our implementation and comparative experimental studies. Moreover, those algorithms demonstrate the feasibility of adapting existing PPDP methods to the PPTP domain, and the challenges in doing so.

The preliminary results of this paper have appeared in [27] (which provides a formal model of the PPTP issue) and [28] (which designs practical PPTP algorithms). However, those previous work share a common limitation in their privacy model, namely, all possible user inputs must be assumed as equally likely to occur, which is usually not the case in real world Web applications. In this paper, we have substantially extended our previous work by addressing this key limitation. Specifically, we re-define the privacy model in Section V-A to accommodate different likelihoods of possible inputs. We then formulate new PPTP problems based on this more realistic privacy model in Section V-B, and we design new algorithms to address several novel challenges in Section V-C. We have also significantly extended the scope of our experimental evaluations in Section VI, by comparing both the previous solutions and our new solutions with more existing methods, on more real world data sets. Finally, we have now provided a formal proof of the intractability of PPTP problems in Section III-B.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

The rest of the paper is organized as follows. Section II formally models the application, privacy, and cost. Section III then employs such models to formulates several PPTP problems, and analyzes the complexity. Section IV devises heuristic algorithms for the formulated problems. Section V proposes an extended version of the PPTP solution, including a re-defined privacy model, the new PPTP problems, and corresponding PPTP algorithms. Section VI discusses the implementation of our solution, and experimentally evaluates the performance of our algorithms. Section VII reviews related work and Section VIII concludes the paper.

II. The PPTP Model

Section II-A first presents the basic model of interaction and observation. Section II-C then maps PPTP to PPDP in order to quantify privacy protection and overhead. Finally, Section II-D extends the basic model to more realistic cases. We will also demonstrate its flexibility to adapt different privacy properties in Section V. Table IV lists main notations that will be used throughout the paper.

a, $\vec{a}$, A_i or A	Action, action-sequence, action-set
s, v, $\vec{v}$, V_i or V	Flow, flow-vector, vector-sequence, vector-set
$\vec{a}[i], \vec{v}[i]$	The i^{th} element in $\vec{a}$ and $\vec{v}$
VA_i or VA	Vector-action set
$pre(a, i)$	i-Prefix
$dom(P)$	Dominant-vector
$vdis(v_1, v_2)$	Vector-distance

TABLE IV
The Notation Table

A. The Basic Model

We model the PPTP issue from two perspectives, the *interaction* between users and servers, and the *observation* made by eavesdroppers. First, Definition 1 formalizes the interaction. Our discussions about Table II demonstrated how one keystroke may affect another in terms of observations (packet sizes), and how an eavesdropper may combine such multiple observations for a refined inference. Such inter-dependent user *actions* are modeled as an *action-sequence* in Definition 1. The concept of *action-set* models a collection of actions whose corresponding observations may be padded together.

Definition 1 (Interaction): Given a Web application, we define

- an *action* a as an atomic user input that triggers traffic, such as a keystroke or a mouse click.
- an *action-sequence* $\vec{a}$ as a sequence of actions with known relationships, such as consecutive keystrokes entered into real-time search engine or a series of mouse clicks on hierarchical menu items. We use $\vec{a}[i]$ to denote the i^{th} action in $\vec{a}$.
- an *action-set* A_i as the collection of all the i^{th} actions in a set of action-sequences. We will simply use A if all action-sequences are of length one.

Example 1: Assume "bee" and "cab" in Table I to be the only possible inputs, we have six actions, a_{11}, a_{12}, a_{13} and a_{21}, a_{22}, a_{23} corresponding to b, e (as second keystroke), e (as third) in input "bee", and c, a, b (as third keystroke) in input "cab". There are two action-sequences $\vec{a}_1 = \langle a_{11}, a_{12}, a_{13}\rangle$ and $\vec{a}_2 = \langle a_{21}, a_{22}, a_{23}\rangle$, and three action-sets $A_1 = \{a_{11}, a_{21}\}$, $A_2 = \{a_{12}, a_{22}\}$, and $A_3 = \{a_{13}, a_{23}\}$. ⊡

Definition 2 models concepts related to the observation made by an eavesdropper. Note a *flow-vector* is only intended to model those packets that may contribute to identify an action and accordingly need to be padded for privacy preservation, such as the s value in Table I (note we are basically making the worst case assumption that adversaries can locate such packets in the traffic (e.g., using de-anonymizing techniques [37]); on the other hand, identifying such packets for deploying a PPTP solution would be relatively easier since the design of a Web application is known). Also, each action is not associated with a flow but a flow-vector, which is itself a sequence, since a single action may trigger more than one packet. Finally, unlike an action-set, is defined as a multiset, since it may contain duplicates (that is, packets may share the same size).

Definition 2 (Observations): Given a Web application, we define

- a *flow* s as the size of a directional packet triggered by a.
- a *flow-vector* v w.r.t. an action a as a sequence of *flows*. Denoted the relation between a and v by $f(a) = v$.
- a *vector-sequence* $\vec{v}$ as a sequence of flow-vectors corresponding to an equal-length action-sequence $\vec{a}$, with each $\vec{v}[i]$ corresponding to $\vec{a}[i]$ $(1 \leq i \leq |\ \vec{v}\ |)$.
- a *vector-set* V_i (or simply V) as the collection of all the i^{th} flow-vectors in a set of vector-sequences, which corresponds to an action-set in the straightforward way.

Example 2: Following Example 1, we have six flow-vectors, $v_{11} = \langle 544\rangle$, $v_{12} = \langle 555\rangle$, $v_{13} = \langle 547\rangle$ and $v_{21} = \langle 554\rangle$, $v_{22} = \langle 560\rangle$, $v_{23} = \langle 558\rangle$ (note that we only model those packets whose sizes can help to identify an action), corresponding to actions a_{11}, a_{12}, a_{13} and a_{21}, a_{22}, a_{23}, respectively. We have two vector-sequences $\vec{v}_1 = \langle v_{11}, v_{12}, v_{13}\rangle$ and $\vec{v}_2 = \langle v_{21}, v_{22}, v_{23}\rangle$, corresponding to action-sequences $\vec{a}_1$ and $\vec{a}_2$, respectively. We have three vector-sets $V_1 = \{v_{11}, v_{21}\}$, $V_2 = \{v_{12}, v_{22}\}$ and $V_3 = \{v_{13}, v_{23}\}$ corresponding to the three action-sets A_1, A_2, and A_3 in Example 1. ⊡

Finally, Definition 3 models the joint information about interaction and observation, which is the collection of the pairs of the action and its corresponding flow-vector.

Definition 3 (Vector-Action Set): Given an action-set A_i and its corresponding vector-set V_i, a *vector-action set* VA_i is the set $\{(v, a) : v \in V_i \wedge a \in A_i \wedge f_i(a) = v\}$.

Example 3: Following above Examples, given the action-set A_1 and vector-set V_1, then the vector-action set is $VA_1 = \{(v_{11}, a_{11}), (v_{21}, a_{21})\}$. Similarly, $VA_2 = \{(v_{12}, a_{12}), (v_{22}, a_{22})\}$, $VA_3 = \{(v_{13}, a_{13}), (v_{23}, a_{23})\}$.

B. Mapping PPDP to PPTP

In the context of privacy-preserving data publishing (PPDP), since the introduction of the k-anonymity concept [34] [38],

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

significant effort has been made on developing efficient algorithms (e.g., [1] [25]), more comprehensive models (e.g., l-diversity [30], t-closeness [26], etc.), semantic privacy notions for resisting adversarial background knowledge (e.g., differential privacy [17]), and so on. Due to space limitations, we only demonstrate the basic ideas of PPDP solutions through an example, and discuss how it may be mapped to privacy-preserving traffic padding (PPTP).

By revisiting the example shown in Table III, we can formulate a classic PPDP problem as follows. We regard the s value as an attribute that is not designed for identifying an individual, but can nonetheless serve this purpose to some extent, namely, a quasi-identifier (e.g., a unique date of birth (DoB)); we regard the user input as an attribute containing sensitive information about individuals, namely, *sensitive value* (e.g., medical conditions); we can then regard this table as to contain medical records of some anonymous patients, released by an insurance company for research purposes. A *linking attack* happens when an adversary re-identifies an individual using the quasi-identifier (DoB), and thus link the individual to his/her sensitive value (medical condition) in the table. The main goal of PPDP is to prevent such linking attacks while still allowing the released data to be useful (e.g., to researchers).

To address this challenge, the k-anonymity model [34] [38] divides the table into *anonymized groups* (as shown in both options in Table III) and then *generalize* the quasi-identifier (DoB) (e.g, by removing the day from a DoB), such that at least k individuals in the table will share the same generalized DoB (with only months and years), and hence a linking attack using this quasi-identifier will fail (since an adversary can not distinguish between the individuals inside an anonymized group). One limitation of this basic model is that privacy cannot be preserved, if all individuals inside an anonymized group happen to have the same medical condition. Therefore, the l-diversity model is introduced to ensure enough diversity (e.g., in its simplest form, at least l different medical conditions) among the sensitive values inside each anonymized group [30]. With such privacy models, the PPDP goal of preventing linking attacks while enabling useful data publication can be modeled as to satisfy a privacy model (e.g., k-anonymity and l-diversity) while maximizing a given utility metric (e.g., minimizing sizes of anonymized groups [1]).

Clearly, as shown in Table III, we can establish the mapping between PPDP and PPTP by regarding the s-value as a quasi-identifier, the user input as a sensitive value, and the padding groups as anonymized groups. Both problems have a similar goal of achieving an optimal tradeoff between privacy (preventing linking attacks in PPDP or side channel attacks in PPTP) and utility (maximizing data utility in PPDP or minimizing padding cost in PPTP). On the other hand, there also exist significant differences between the two. As an example, in Table III, the second option will likely be considered as worse than the first in the PPDP domain in terms of typical data utility measures (intuitively, the second option leads to more utility loss due to its larger anonymized groups), whereas it is actually better in the PPTP domain with respect to padding cost (it can be shown that the second option incurs totally 24 bytes of overhead, in contrast to 33 by the first option). As another example, the effect of combining two keystrokes will be equivalent to releasing multiple inter-dependent tables, which actually leads to a novel PPDP problem (detailed later).

C. Privacy and Cost Model

We now devise the privacy model of PPTP based on the mapping discussed in the previous section. For simplicity, we first consider a simplified case where every action-sequence and flow-vector are of length one, namely, the Single-Vector Single-Dimension (SVSD) case. That is, all actions are independent, and each action triggers only a single packet that can be used to identify the action. In this case, we map a given vector-action set $VA=\{(v,a) : v \in V \wedge a \in A \wedge f(a)=v\}$ to a table $T(v,a)$ with two attributes, the flow-vector v (equivalent to a flow s here) as quasi-identifier and the action a as sensitive attribute. Note that we will interchangeably refer to a vector-action set and its tabular representation from now on.

Definition 4 quantifies the amount of privacy protection under a given vector-action set. This model follows the widely adopted approach of assuming a fixed privacy requirement while minimizing the cost.

Definition 4 (k-Indistinguishability): Given a vector-action set VA, we define

- a *padding group* as any $S \subseteq VA$ satisfying that all the pairs in S have identical flow-vectors and no $S' \supset S$ can satisfy this property, and
- we say VA satisfies *k-indistinguishability* (k is an integer) or VA is *k-indistinguishable* if the cardinality of every padding group is no less than k.

Discussion One may argue that, in contrast to encryption, *k-indistinguishability* may not provide strong enough protection. However, as mentioned before, we are considering cases where encryption is already broken by side-channel attacks, so the strong confidentiality provided by encryption is already not an option. Moreover, in theory k could always be made sufficiently large to provide enough confidentiality, although a reasonably large k would usually satisfy users' privacy requirements for most practical applications. Finally, since most web applications are publicly accessible and consequently an eavesdropper can unavoidably learn about the list of all possible inputs, we believe perfect confidentiality is not always achievable. Another subtlety here is that padding packets to be k-indistinguishable does not leak information to adversaries about which packets are sensitive (padded), because they can only see padded packets but not the original ones.

Although the PPTP privacy model is adapted from its counterpart in PPDP, the former has a unique characteristic, that is, the sensitive values (actions) are always unique, and hence by satisfying k-indistinguishability, the vector-action set also satisfies *l-diversity* ($l=k$) in its simplest form [30]. That is, each padding group has at least k different actions, so no eavesdropper can determine which of the k (or more) actions has triggered an observed flow-vector. This simple model is sufficient to demonstrate the usefulness of mapping PPTP to PPDP. For simplicity, we will focus on *k-indistinguishability* in Sections II– IV, and delay the discussion about applying more general forms of l-diversity in Section V to address cases where not all actions should be treated equally in padding.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

In addition to privacy requirement, we also need a quantitative measure for the cost of padding and processing. Across the whole vector-set, Definition 5 counts the number of additional bytes after padded, while Definition 6 counts the number of flows that are involved in padding. We focus on these simple models in this paper while there certainly exist other ways for modeling such costs.

Definition 5 (Distance and Padding Cost): Given a vector-set V, we define

- the *vector-distance* between two equal-length flow-vectors v_1 and v_2 as: $vdis(v_1, v_2) = \sum_{i=1}^{|v_1|}(|s_{1i} - s_{2i}|)$ where s_{1i} and s_{2i} are the i^{th} flow in v_1 and v_2, respectively.
- the *padding cost* of V as: $cost = \sum_{i=1}^{|V|}(vdis(v_i, v_i'))$ where v_i and v_i' denote a flow-vector in V and its counterpart after padding, respectively.

Definition 6 (Processing Cost): Given a vector-set V, we define the processing cost of V as the number of flows in V which corresponding packets should be padded.

D. The SVMD and MVMD Cases

In the previous section, we have focused on the simplified SVSD case to facilitate a focused discussion on the privacy and cost model. We now look at the more realistic cases. First, we consider the Single-Vector Multi-Dimension (SVMD) case where each flow-vector may include more than one flow (that is, an action may trigger more than one packet that can be used to identify the action), whereas each action-sequence is still composed of a single action. In this case, the vector-action set needs to be mapped to a table $T(s_1, \ldots, s_{|v|}, a)$ with multiple quasi-identifier attributes (each flow corresponds to an attribute). Thus, based on Definition 4, flow-vectors can form a padding group only if they are identical with respect to every flow inside the vectors. Another subtlety is that the model of vector-action set requires all the flow-vectors to have the same number of flows, which is not always possible in practice. One solution is to insert dummy packets of size zero which will then be handled as usual in the process of padding.

Next, we consider the Multi-Vector Multi-Dimension (MVMD) case in which each action-sequence consists of more than one actions and each flow-vector includes multiple flows. Definition 7 expresses the relationship between actions in an action-sequence.

Definition 7 (i-prefix, adjacent-prefix, adjacent-suffix): We define

- the *i-prefix* of an action-sequence $\vec{a} = \langle \vec{a}[1], \vec{a}[2], \ldots, \vec{a}[t] \rangle$ ($i \in [1, t]$), denoted as $pre(\vec{a}, i)$, as the sequence $\langle \vec{a}[1], \vec{a}[2], \ldots, \vec{a}[i] \rangle$, and we say $\vec{a}[i-1]$ is the *adjacent-prefix* (or simply prefix) of $\vec{a}[i]$, and $\vec{a}[i+1]$ is the *adjacent-suffix* (or simply suffix) of $\vec{a}[i]$,
- similarly, we define the *i-prefix* of vector-sequence $\vec{v}$, and the *prefix, suffix* of $\vec{v}[i]$.

In the MVMD case, due to the prefix relationship, the flow-vector for an action may provide additional information about flow-vectors that correspond to the previous actions in the same action-sequence. Such knowledge may enable the eavesdropper to refine his guesses about an action. Such a scenario is illustrated in Figure 1. Also, we slightly change the definition of a vector-action set to accommodate the added prefix action information, as shown in Definition 8. We will delay the discussion about how a padding algorithm may satisfy k-indistinguishability in this case to the next section.

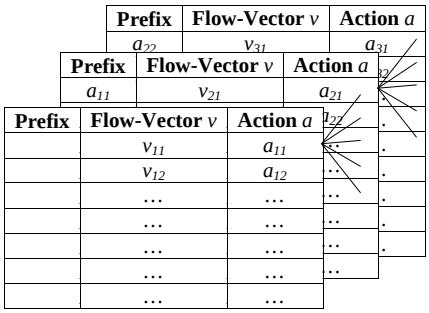


Fig. 1. The Vector-Action Set in MVMD Case

Definition 8 (Vector-Action Set (MVMD Case)): Given t action-sets $\{A_i : 1 \leq i \leq t\}$ and the corresponding vector-sets $\{V_i : 1 \leq i \leq t\}$, the *vector-action set* VA is the collection of sets $\{\{(v, a) : v \in V_i \wedge a \in A_i \wedge f_i(a) = v\} : 1 \leq i \leq t\}$.

III. The PPTP Problems and Complexity

The formal model introduced in the previous section enables us to formulate a series of PPTP problems and study their complexity. We first discuss the choice of our ceiling padding approach among other possibilities in Section III-A, and then address the SVSD and SVMD cases in Section III-B and the MVMD case in Section III-C.

A. Ceiling Padding

In choosing a padding method, we need to address two aspects, privacy protection by satisfying the k-indistinguishability property, and minimizing padding cost. As previously mentioned, an application-agnostic approach, such as packet-size rounding and random padding, will usually incur high padding cost while not necessarily guaranteeing sufficient privacy protection [14]. We now revisit this argument by showing that a larger rounding size does not necessarily lead to more privacy. With our model, more privacy can now be clearly defined as satisfying k-indistinguishability for a larger k. Consider rounding the flows for the second keystrokes shown in Table II to a multiple of 64 (for example, 487 to $8 \times 64 = 512$). It can be shown that such rounding can achieve 2-indistinguishability (detailed calculations are omitted), while increasing the rounding size to 160 can achieve 3-indistinguishability. However, further increasing it to 256 can still only satisfy 2-indistinguishability.

From another point of view, as demonstrated in Section II-B, we can now apply the PPDP technique of generalization to addressing the PPTP problem. Recall that a generalization technique will partition the vector-action set into groups, and then break the linkage among actions in the same group. One unique aspect in applying generalization to PPTP is that padding can only increase each packet size but cannot decrease or replace it with a range of values like in normal generalization. The above considerations lead to a new padding

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

method given in Definition 9. Basically, after partitioning a vector-action set into groups, we pad each flow in a padding group to be the maximum size of that flow in the group.

Definition 9 (Dominance and Ceiling Padding): Given a vector-set V, we define

- the *dominant-vector* $dom(V)$ as the flow-vector in which each flow is equal to the maximum of the corresponding flow among all the flow-vectors in V.
- a *ceiling-padded* group in V as a padding group in which every flow-vector is padded to the dominant-vector. We also say V is ceiling-padded if all the groups are ceiling-padded.

Similar to the centroid in *k-means clustering* [24], dominant-vector is not necessary to be an actual vector in V. We will focus on the ceiling padding method in the rest of the paper. When no ambiguity is possible, we will not distinguish between vector-set, vector-action set, flow-vector, and vector-sequence.

B. The SVSD and SVMD Cases

In the SVSD case, there is only a single flow in each flow-vector of the vector-set. Therefore, we only need to modify the vector-set by increasing the value of some flows to form padding groups. The padding problem can be formally defined as follows.

Problem 1 (SVSD Problem): Given a vector-action set VA and the corresponding vector-set V and action-set A, the privacy property $k \leq |V|$, find a partition P^{VA} on VA such that the corresponding partition on V, denoted as $P^V = \{P_1, P_2, \ldots, P_m\}$, satisfies

- $\forall(i \in [1, m]), |P_i| \geq k$;
- The padding cost $\sum_{i=1}^{m}(dom(P_i) \times |P_i|)$ is minimal. ⊡

In the SVMD case, there are more than one flows in each flow-vector of the vector-set. The padding problem can be defined as follows:

Problem 2 (SVMD problem): Given a vector-action set VA and the corresponding vector-set V (in which each flow-vector includes n_p flows) and action set A, the privacy property $k \leq |V|$, find a partition P^{VA} on VA such that the corresponding partition on V, denoted as $P^V = \{P_1, P_2, \ldots, P_m\}$, satisfies

- $\forall(i \in [1, m]), |P_i| \geq k$;
- The cost $\sum_{i=1}^{m}(\sum_{j=1}^{n_p}((dom(P_i))[j]) \times |P_i|)$ is minimal. ⊡

Theorem 1 shows that the above PPTP problem is intractable, and indicates that Problem 2 is NP-hard even when there are only two different flow values in the vector-set.

Theorem 1: Problem 2 is NP-complete when $k=3$ and the flow-vectors are from binary alphabet $\sum$.

Proof: The proof follows the work in [1] for reduction from the well-known NP-hard problem, namely, the problem of Edge Partition Into Triangles (EPIT) [23], which is defined as follows:

Given a graph $G = (V, E)$ with $|E| = 3m$ for some integer m, can the edges of G be partitioned into m triangles with disjoint edges? This problem is still NP-hard even G is simple.

Reduction of EPIT to Problem 2: For an instance of EPIT problem (w.l.o.g., the graph is assumed to be simple), we construct a vector-action set VA with $3m$ pairs of (vector, action) where the vector has $|V|$ number of flows. Concretely, for each edge $u_iu_j \in E$, to which u_i and u_j are incident, we create a (v, a) pair in VA such that the two flows in v corresponding to u_i and u_j have z_b's and all the other flows have z_a's (each vertex is bijectively mapped to a flow in the flow-vector denoted by the subscript), where z_a and z_b are two positive integers and $z_a < z_b$. Obviously, this reduction works in polynomial time $O(3mn)$. Note that padding is represented here by modifying z_a's in some flows to be z_b's, and correspondingly, the cost is shown by the total number of z_a's which is changed to z_b's. We show that the cost of optimal solution for Problem 2 is at most $3m$ if and only if E in G can be partitioned into a collection of m disjoint triangles.

From EPIT to Problem 2: Suppose that there exists a partition of edges in G for EPIT. Consider any triangle with vertexes u_i,u_j and u_l, and edges u_iu_j, u_ju_l and u_lu_i. Observe that, by modifying the l, i, and j flows from z_a to z_b respectively in the corresponding (v, a) pairs, we obtain a group in size 3 with identical flow-vectors. Consequently, we get a solution to Problem 2 with cost $3m$.

From Problem 2 to EPIT: Suppose that there is a 3-indistinguishability solution for Problem 2 of cost at most $3m$. Since G is simple, any two pairs in VA are different in at least 2 flows (in the case that the corresponding edges in G share one common vertex). Consequently, to make two pairs have identical flow-vector values, we should at least modify one flow from z_a to z_b for each (v, a) pair. Therefore, the solution cost is exactly $3m$ and each pair is padded exactly one flow.

Since the solution satisfies 3-indistinguishability, any group in VA should be in size at least 3. Observe that, for any group with size larger than 3, any of its pairs will have at least two flows which are z_a and at least one other pair in the group has value z_b. In other words, in any such group, each pair should be at least padded two flows. Thus, each group has exactly three pairs. The only possibility is that the corresponding edges for the tuples in each group composes a triangle.

Besides, a given solution of Problem 2 can be verified in polynomial time whether it satisfies k-indistinguishability and the cost is less than a given value or not. ■

Note that, at first glance, the SVMD problem may resemble the problem of *k-means clustering* [24]. However, algorithms for *k-means clustering* cannot be directly applied to our problem due to following differences between these two problems. First, *k-means clustering* needs to partition a multiset into k groups, whereas in our problem, the minimal size of each group must be at least k. Second, *k-means clustering* is to minimize the within-cluster sum of squares, while our problem is to minimize the total distance between each of the flow-vectors and the dominant-vector.

C. MVMD Problem

As mentioned in Section II-D, by correlating flow-vectors in the vector-sequence, an eavesdropper may refine his guesses of the actual action-sequence. We first discuss the challenges of traffic padding in such cases by observing the traffic for the sequence of two keystrokes as shown in Table II.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

Example 4: To revisit Table II, suppose an eavesdropper has only observed the flow for the second keystroke. In order to preserve 2-indistinguishability, one algorithm may partition the 16 cells into 8 groups of size 2 and assume that the queried strings $(a)c$ and $(c)a$ form one group. When the eavesdropper observes that the flow for the second keystroke is 501, she cannot determine whether the queried string is $(a)c$ or $(c)a$.

However, suppose the eavesdropper also observes the flow for the first keystroke, she can determine that the first keystroke is either (a) or (c) when the flow is 509 or 502, respectively. Consequentially, she can infer the queried string by combining these two observations. ⊡

One seemingly valid solution is padding the flow-vectors to satisfy 2-indistinguishability for each keystroke separately. Unfortunately, this will fail when correlating two consecutive observations. The reason is as follows. To pad traffic for the first keystroke, the optimal solution is to partition $(a)-(d)$ into two groups, $\{(b),(c)\}$ and $\{(a),(d)\}$. However, when the eavesdropper observes the flow of first keystroke, she can still determine it must be either (a) or (c) when the size is 516 or 504, respectively, because only when the first keystroke starts with (a) or (c) can the flow for second keystroke be padded to 501. Thus, the eavesdropper will eliminate (b) and (d) from possible guesses, which violates 2-indistinguishability.

Another seemingly viable solution is to first collect the vector-sequences for all input strings and then pad them such that an input string as a whole cannot be distinguished from at least $k-1$ others. Unfortunately, such an approach cannot ensure the privacy, either. For example, one algorithm may split $(a)c$ and $(a)b$ into two different groups, where (a) should be padded to 509 and 516, respectively. When the server receives (a), it must immediately respond due to auto-suggestion feature. However, since the server cannot predict the next keystroke will be b or c (worse put, or others), it cannot decide whether to pad (a) to 509 or 516.

The discussed challenges mainly arise due to the approach of padding each vector-set independently. We now propose a different approach. Intuitively, the partitioning of a vector-set corresponding to each action will *respect* the partitioning results of all the previous actions in the same action-sequence. More precisely, the padding of different vector-sets is correlated based on the following two conditions.

- Given two t-sized vector-sequences $\vec{v}_1$ and $\vec{v}_2$, any prefix $pre(\vec{v}_1,i)$ and $pre(\vec{v}_2,i)$ $(i \in [2,t])$, can be padded together only if $\forall(j<i)$, $pre(\vec{v}_1,j)$ and $pre(\vec{v}_2,j)$ are padded together.
- For any two t-sized action-sequences $\vec{a}_1$ and $\vec{a}_2$ and corresponding vector-sequences $\vec{v}_1$ and $\vec{v}_2$, if $pre(\vec{a}_1,i) = pre(\vec{a}_2,i)(i \in [1,t])$, then $pre(\vec{v}_1,i)$ and $pre(\vec{v}_2,i)$ must be padded together.

Once a partition satisfies these conditions, no matter how an eavesdropper analyzes traffic, either for an action alone or combining multiple observations of previous actions, the mental image about actual action-sequence remains the same.

Following Example 4, with these conditions, $(a)c$ and $(c)a$ can form a group only if their prefixes (a) and (c) are in same group. This ensures $(a)c$ and $(c)a$ always have same flow values in the sequence.

Due to the similarity between the conditions and a related concept in graph theory, we call a partition satisfying such conditions the *oriented-forest partition*.

Problem 3 (MVMD problem): Given a vector-action set $VA = (VA_1, VA_2, \ldots, VA_t)$ where $VA_i = (V_i, A_i)$ $(i \in [1,t])$, the privacy property $k \leq |V_t|$, find the partition P^{VA_i} on VA_i such that the corresponding partition $P^{V_i} = \{P_1^i, P_2^i, \ldots, P_{m_i}^i\}$ on V_i satisfies

- $\forall((i \in [1,t-1]) \wedge (j \in [1,m_i]))$
  $\begin{cases} |P_j^i| \geq k, & \text{if } (|V_i| \geq k), \\ |P_1^i| = |V_i|, & \text{if } (|V_i| < k); \end{cases}$
- $\forall(j \in [1,m_t]), |P_j^t| \geq k$;
- The sequence of P^{V_i} is an oriented-forest partition;
- The total padding cost of P^{V_i} $(i \in [1,t])$ is minimal. ⊡

Obviously, Problem 3 is also NP-complete when $k \geq 3$ since Problem 2 is special case of Problem 3.

IV. The Algorithms

As discussed above, our main idea is to partition the vector-action set into padding groups, and then break the linkage inside each group through padding to satisfy a given privacy as well as minimize the cost. In this section, we design three heuristic algorithms to demonstrate the existence of abundant possibilities in approaching this PPTP issue. Note that when the cardinality of vector-action set is less than the privacy property k, there is no solution to satisfy the privacy property. In such cases, our algorithms will simply exit, which will not be explicitly shown in each algorithm hereafter.

A. The svsdSimple Algorithm

The main intention of presenting the svsdSimple algorithm is to show that, when applying k-indistinguishability to PPTP problems, an algorithm may sometimes be devised in a very straightforward way, and yet achieve a dramatic reduction in costs when compared to existing approaches (as shown in the Section VI). Basically, the svsdSimple algorithm attempts to minimize the cardinality of padding groups in the SVSD case (refer to [28] for detail).

B. The svmdGreedy Algorithm

The svmdGreedy algorithm, which aims at both SVSD and SVMD problems, is shown in Table V. Roughly speaking, the svmdGreedy recursively divides the padding group P_i in P^{VA}, where $|P_i| \geq 2 \times k$, into two padding groups P_{i1} and P_{i2} until the cardinality of any padding group in P^{VA} is less than $2 \times k$. When svmdGreedy splits a padding group $P_i(VA_i)$ into two, these resultant padding groups, P_{i1} and P_{i2}, must satisfy that $(P_{i1} \cup P_{i2} = P_i) \wedge (P_{i1} \cap P_{i2} = \emptyset) \wedge (|P_{i1}| \geq k) \wedge (|P_{i2}| \geq k)$. Obviously, there must exist many solutions of P_{i1} and P_{i2}. svmdGreedy limits the optimizing process insides a subset of possible solutions as follows. For each flow, svmdGreedy first sorts the flow-vectors in non-decreasing order of that flow, then splits P_i into P_{i1} and P_{i2} at position pos in the sorted sequence where $(pos \in [k, |P_i|-k])$. There are totally $(n_p \times (|P_i| - 2 \times k))$ possible solutions for all flows in the flow-vector, where n_p is the number of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

flows in flow-vector. SvmdGreedy finally selects the one with minimal padding cost among this set of solutions. Clearly, this algorithm can solve SVSD-problem when n_p is set to be 1.

Algorithm svmdGreedy
Input: a vector-action set VA, the privacy property k;
Output: the partition P^{VA} of VA;
Method:
1. **If**($|VA| < 2 \times k$)
2. Create in P^{VA} the VA;
3. **Return**;
4. **Let** n_p be the number of flows in flow-vector;
5. **For** $p = 1$ to n_p
6. **Let** S_p^{VA} be the sequence of VA in the non-decreasing order of the p^{th} flow in the flow-vector;
7. **For** $i = k$ to $|S_p^{VA}| - k$
8. **Let** $cost_{p,i}$ as the cost when S_p^V is split at position i;
9. **Let** $cost_p$ be a pair (c, i) where c is the minimal in $(cost_{p,i})$ and i is the corresponding position;
10. **Let** $cost$ be a triple (c, p, i) where c is the minimal in c of $cost_p(p \in [1, n_p])$, and p and i are the corresponding p and i;
11. Split $S_{cost.p}^{VA}$ into VA_1 and VA_2 at position $cost.i$;
12. **Return** svmdGreedy(VA_1);
13. **Return** svmdGreedy(VA_2);

TABLE V
THE SVMDGREEDY ALGORITHM FOR SVMD-PROBLEM

The svmdGreedy algorithm has an $O(n_p \times n^2)$ time complexity in the worst case (each time, the algorithm splits P_i into k-size P_{i1} and $(|P_i| - k)$-size P_{i2}), and $O(n_p \times n \times logn)$ in average cases (each time, the algorithm halves P_i), where $n = |VA|$.

C. The mvmdGreedy Algorithm

Both svsdSimple and svmdGreedy algorithms tackle cases where each action-sequence consists of a single action (correspondingly, each vector-sequence consists of a single flow-vector). Our intention now in devising the mvmdGreedy is to demonstrate how the two conditions mentioned in Section III-C facilitate the algorithm design. In this algorithm, we extend PPDP solutions to a sequence of inter-dependent vector-action sets. The only constraint in partitioning vector-action set VA_i is to ensure all flow-vectors in a padding group should have their prefix in an identical padding group of VA_{i-1}.

The mvmdGreedy algorithm for MVMD-Problem is shown in Table VI. Roughly speaking, mvmdGreedy partitions each vector-action set in the sequence in the given order, each for the flow-vector corresponding to an action in an action-sequence. More specifically, mvmdGreedy applies svmdGreedy to partition the first vector-action set in the sequence. For each remaining vector-action set VA_i, mvmdGreedy first partitions it into $|P^{VA_{i-1}}|$ number of padding groups based on the adjacent-prefix of the flow-vectors, and then applies svmdGreedy to further partition these padding groups.

Similarly, the mvmdGreedy also has an $O(n_p \times n^2)$ time complexity in the worst case (each time, the algorithm splits VA_i into k-size VA_{i1} and $(|VA_i| - k)$-size VA_{i2}), and $O(n_p \times n \times logn)$ in average cases (each time, the algorithm halves VA), where n is the total number of flow-vectors in those vector-sets.

Algorithm mvmdGreedy
Input: a t-size sequence D of vector-action sets, the privacy property k;
Output: the partition P^D of D;
Method:
1. **Let** $D = (VA_1, VA_2, \ldots, VA_t)$;
2. **Let** $P^1 = svmdGreedy(VA_1, k)$;
3. **For** each $(w \in [1, |P^1|])$, assign group $G_w^1 \in P^1$ a unique $gid = w$;
4. **For** $i = 2$ to t
5. Create in P^i $|P^{i-1}|$ number of empty groups $G_w^i (w \in [1, |P^{i-1}|])$;
6. **For** each v_{ia} in VA_i
7. **Let** w be the gid of the group G_w^{i-1} in P^{i-1} that the prefix of v_{ia} in VA_{i-1} belongs to;
8. Insert v_{ia} into G_w^i;
9. **For** each $(w \in [1, |P^{i-1}|])$
10. $P^i = (P^i \setminus G_w^i) \cup svmdGreedy(G_w^i, k)$;
11. **For** each $(w \in [1, |P^i|])$, assign group $G_w^i \in P^i$ a unique $gid = w$;
12. **Return** $P^D = \{P^i : 1 \leq i \leq t\}$;

TABLE VI
THE MVMDGREEDY ALGORITHM FOR MVMD-PROBLEM

V. EXTENSION TO L-DIVERSITY

In previous discussion, we implicitly assume that each action in an action-set is equally likely to occur. However, in real life, each action is not necessary to have equal probability to be performed. In this section, we discuss an extension to our model to further demonstrate that many existing PPDP concepts may be adapted to address PPTP issues. Specifically, we adapt the l-diversity [30] concept to address cases where not all actions should be treated equally in padding (for example, some statistical information regarding the likelihood of different inputs may be publicly known).

A. The Model

We first assign an integer *weight* to each action to catch the information about its *occurrence probability* among the action-set that it belongs to. The reason for assigning *weight* instead is due to the utilization of such as *access counter* for visit statistics in most applications.

Definition 10 (Weight-Set): Given an action-set A_i, the weight-set W_i is defined as the collection of integer weights associated with the actions in that action-set.

Definition 11 (Occurrence Probability): Given an action-set A and corresponding weight-set W, the *occurrence probability* of an action a with weight w in A is defined as $pr(a, A) = \frac{w}{\sum_{i=1}^{|W|}(w_i)}$.

Example 5: To revisit Example 1, given the action-set $A_1 = \{a_{11}, a_{21}\}$, assume that the weight for $a_{11} = b$ and $a_{21} = c$ are 20 and 5 respectively (clearly, in practice the value of weight should be assigned based on the characteristics of applications). Then, the weight-set is $W_1 = \{20, 5\}$. Moreover, in action-set A_1, the occurrence probability of b and c is $\frac{20}{20+5} = 80\%$ and $\frac{5}{20+5} = 20\%$, respectively. ⊡

Then we slightly change the definition of vector-action set to accommodate the weight information. Since SVSD and SVMD are special cases of MVMD, w.l.o.g., we only redefine the concept for MVMD.

Definition 12 (Vector-Action-Weight Set): Given t action-sets $\{A_i : 1 \leq i \leq t\}$, and the corresponding weight-sets $\{W_i : 1 \leq i \leq t\}$ and vector-sets $\{V_i : 1 \leq i \leq t\}$,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

the *vector-action-weight set* VAW is the collection of sets $\{\{(v, a, w) : v \in V_i \wedge a \in A_i \wedge w \in W_i\} : 1 \leq i \leq t\}$.

Example 6: Following Example 5, given the action-set $A_1 = \{b, c\}$, weight-set $W_1 = \{20, 5\}$ and vector-set $V_1 = \{544, 554\}$, the corresponding vector-action-weight set is $VAW_1 = \{(544, b, 20), (554, c, 5)\}$. ⊡

We regard weight information as an additional information about the action, therefore, the mapping from PPTP to PPDP is consistent with what has been discussed before. Definition 13 applies *l-diversity* to quantify the amount of privacy protection under a given vector-action-weight set.

Definition 13 (l-Diversity): Given a vector-action set VAW, we define

- a *padding group* as any $S \subseteq VAW$ satisfying that all the pairs in S have identical flow-vectors and no $S' \supset S$ can satisfy this property, and
- we say VAW satisfies *l-diversity* (l is an integer) or VAW is l-diverse if the *occurrence probability* of any action in any padding group is no greater than $\frac{1}{l}$.

Example 7: Following Example 6, the highest occurrence probability is b with $\frac{4}{5}$. Since $\frac{1}{1} > \frac{4}{5} > \frac{1}{2}$, VAW_1 does not satisfy 2-diversity. ⊡

B. The Problem

With the revised definitions, we now formulate the diversity problems, namely, SVSD-Diversity, SVMD-Diversity, and MVMD-Diversity, for the SVSD case, SVMD case, and MVMD case, respectively. Clearly, the main difference between the l-diversity problems and aforementioned k-indistinguishability problems is the condition on the padding group. That is, for k-indistinguishability, the cardinality of each padding group should be at least k, whereas, for l-diversity, the maximal occurrence probability of each group should be at most $\frac{1}{l}$, as demonstrated by Problem 4 for MVMD case.

Problem 4 (MVMD-Diversity Problem): Given a vector-action-weight set $VAW = (VAW_1, VAW_2, \ldots, VAW_t)$ where $VAW_i = (V_i, A_i, W_i)$ $(i \in [1, t])$, the privacy property $l \leq \frac{1}{\max\limits_{a \in A_t}(pr(a, A_t))}$, find the partition P^{VAW_i} on VAW_i such that the corresponding partitions $P^{A_i} = \{P_1^{A_i}, P_2^{A_i}, \ldots, P_{m_i}^{A_i}\}$ on A_i and $P^{V_i} = \{P_1^{V_i}, P_2^{V_i}, \ldots, P_{m_i}^{V_i}\}$ on V_i satisfy

- $\forall((i \in [1, t-1]) \wedge (j \in [1, m_i]))$
$$\begin{cases} \max\limits_{a \in P_j^{A_i}}(pr(a, P_j^{A_i})) \leq \frac{1}{l}, & \text{if } (\max\limits_{a \in A_i}(pr(a, A_i)) \leq \frac{1}{l}), \\ P_1^{A_i} = A_i, & \text{if } (\max\limits_{a \in A_i}(pr(a, A_i)) > \frac{1}{l}); \end{cases}$$
- $\forall(j \in [1, m_t]),\ \max\limits_{a \in P_j^{A_t}}(pr(a, P_j^{A_t})) \leq \frac{1}{l}$;
- The sequence of P^{V_i} is an oriented-forest partition;
- The total padding cost of P^{V_i} $(i \in [1, t])$ after applying ceiling padding is minimal. ⊡

Observe that when the weights of all actions in any VAW_i are set to be identical, and $l = k$, Problem 4 is simplified to Problem 3. Informally, Problem 4 is at least as hard as Problem 3.

Although l-diversity in PPTP shares the same spirit with that in PPDP, algorithms for l-diversity in PPDP cannot be directly applied to our PPTP problems due to the following main difference between these two problems. In PPDP, there are many tuples with same sensitive values in the micro-data table, while in our problem, the action in an action-set is not duplicated, and we assign a weight for each action to distinguish its possibility to be performed by a user from other actions.

C. The Algorithms

To facilitate the explanation, we first present the svsdDiversity algorithm for SVSD case to show the essence of design ideas to satisfy l-diversity as shown in Table VII. Roughly speaking, svsdDiversity algorithm first sorts the actions in non-increasing order of their *weight* values, and then among the actions with same *weight*, sorts them in a predefined order based on their flow-vectors. In this algorithm, we sort them in non-increasing order of the flows (note this step aims at reducing the padding cost in the resultant group and there must exist alternative solutions for ordering). Based on the sorted version S of vector-action-weight set, svsdDiversity iteratively removes actions from S to construct the padding group until S is empty. In each iteration, svsdDiversity splits the sequence S into two l-diverse sub-sequences, $P_{\alpha-}$ and $P_{\alpha+}$, such that the first sub-sequence $P_{\alpha-}$ has minimal possible cardinality.

Algorithm svsdDiversity
Input: a vector-action-weight set VAW, the privacy property l;
Output: the partition P^{VAW} of VAW;
Method:
1. **Let** $P^{VAW} = \emptyset$;
2. **Let** S be the sequence of VAW in a non-increasing order of its W;
3. **If**$(pr(S[1], S) > \frac{1}{l})$
4. **Return**;
5. **Sort** elements in S with same *weight* value in non-increasing order of its V;
6. **While** $(S \neq \emptyset)$
7. **Let** $P_{\alpha-} = \{S[i] : i \in [1, \alpha]\}$, $P_{\alpha+} = \{S[i] : i \in [\alpha + 1, |S|]\}$;
8. **Let** $\alpha \in [l, |S|]$ be the smallest value such that: $pr(S[1], P_{\alpha-}) \leq \frac{1}{l}$ and $(pr(S[\alpha + 1], P_{\alpha+}) \leq \frac{1}{l}$ or $P_{\alpha+} \equiv \emptyset)$
9. Create partition $P_{\alpha-}$ on P^{VAW};
10. $S = P_{\alpha+}$;
11. **Return** P^{VAW};

TABLE VII
THE SVSDDIVERSITY ALGORITHM FOR SVSD-DIVERSITY CASE

Note that, in each iteration, the algorithm removes $P_{\alpha-}$ from S and further splits $P_{\alpha+}$. Before discussing the reasons, we first introduce the *undividable diverse group* concept to define the padding group which can not be further split without reordering the sequence.

Definition 14 (Undividable Diverse Group): Given a vector-action-weight set VAW, and denote by $S = (S[1], S[2], \ldots, S[|A|])$ the sequence of VAW in the non-increasing order of its W, we say $P_{\alpha-} = (S[1], S[2], \ldots S[\alpha])$, a sub-sequence of S, is a *undividable diverse group*, if

- $pr(S[1], P_{\alpha-}) \leq \frac{1}{l}$, and
- there does not exist any integer $\beta \in [1, \alpha)$, such that both $pr(S[1], (S[1], \ldots, S[\beta])) \leq \frac{1}{l}$ and $pr(S[\beta + 1], (S[\beta + 1], \ldots, S[\alpha])) \leq \frac{1}{l}$ hold.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

The $P_{\alpha-}$ in step 9 is a undividable diverse group by reasoning as follows. If α is the smallest position that $P_{\alpha-}$ satisfies l-diversity, clearly, it cannot be further split. Otherwise, suppose that β is the smallest position such that $\beta < \alpha$ and $P_1 = (S[1], S[2], \ldots, S[\beta])$ satisfies l-diversity, then $P_2 = (S[\beta+1], \ldots, S[\alpha])$ is not l-diverse, since based on the condition in step 8, $pr(S[\beta+1], P_2) = \frac{w[\beta+1]}{\sum_{i=\beta+1}^{\alpha} w[i]} \geq \frac{w[\beta+1]}{\sum_{i=\beta+1}^{|S|} w[i]} > \frac{1}{l}$. Similarly, splitting $P_{\alpha-}$ at any position between β and α leads to same result, which confirms the statement.

Furthermore, svsdDiversity always terminates since appending action with smaller weight value to an l-diverse padding group will never produce a group violating l-diversity. Therefore, each iteration will result in either two l-diverse groups or one whole sequence together with an empty sequence.

The svsdDiversity algorithm has $O(nlogn)$ time complexity since step 2 and step 5 cost $(nlogn)$ time and each action is considered once for the remaining steps, where $n = |VAW|$.

Then, we design svmdDiversity and mvmdDiversity algorithms for SVMD-Diversity and MVMD-Diversity problems, respectively. Similar to svsdDiversity, svmdDiversity follows the conditions shown in step 8 in Table VII to split S as shown in Table VIII. In contrast to svsdDiversity, svmdDiversity first identifies all possible positions of given VAW which satisfy the conditions for each flow, and then selects the one with minimal cost among all possible positions in all the flows. The svmdDiversity has $O(n_p \times n^2)$ in the worst case and $O(n_p \times n \times logn)$ in average cases.

Algorithm svmdDiversity
Input: a l-diverse vector-action-weight set VAW, the privacy property l;
Output: the partition P^{VAW} of VAW;
Method:
1. **If**($|VAW| < 2 \times l$)
2. Create in P^{VAW} the VAW;
3. **Return**;
4. **Let** S be the sequence of VAW in a non-increasing order of its W;
5. **Let** n_p be the number of flows in flow-vector V;
6. **For** $p = 1$ to n_p
7. **Sort** elements in S with same *weight* value in non-increasing order of the p^{th} flow in its V;
8. **Let** $P_{\alpha_{p,i}-} = \{S[r] : r \in [1, \alpha_{p,i}]\}$, and $P_{\alpha_{p,i}+} = \{S[r] : r \in [\alpha_{p,i}+1, |S|]\}$;
9. **Let** $Z_p \subseteq \{i : l \leq i \leq |S|\}$ be the set of values such that: $\forall(\alpha_{p,i} \in Z_p)$, $pr(S[1], P_{\alpha_{p,i}-}) \leq \frac{1}{l}$ and $(pr(S[\alpha_{p,i}+1], P_{\alpha_{p,i}+}) \leq \frac{1}{l}$ or $P_{\alpha_{p,i}+} \equiv \emptyset)$
10. **Let** α_p be the value in Z_p such that the cost is minimal among all $\alpha_{p,i} \in Z_p$ when S is split at $\alpha_{p,i}$;
11. **Let** $\alpha \in \{\alpha_p : p \in [1, n_p]\}$ with the minimal cost;
12. **If** ($P_{\alpha+}$ is empty)
13. Create in P^{VAW} the $P_{\alpha-}$;
14. **Return**;
15. **Return** svmdDiversity($P_{\alpha-}$);
16. **Return** svmdDiversity($P_{\alpha+}$);

TABLE VIII
THE SVMDDIVERSITY ALGORITHM FOR SVMD-DIVERSITY CASE

Similar to mvmdGreedy, mvmdDiversity first ensures that the partitioning satisfies the conditions of an oriented-forest partition (refer to Section IV-C for the main idea). There is still another complication. In mvmdGreedy, an initial padding group based on prefixes certainly satisfies k-indistinguishability only if the set of prefixes satisfies. However, this is not the case in mvmdDiversity since a vector-action set with size larger than l will not necessarily be a l-diverse set. To address this issue, we confine the Z_p in step 9 of svmdDiversity in Table VIII to further satisfy that both the set formed by all suffixes (refer to Definition 7) of $P_{\alpha_{p,i}-}$ and that of $P_{\alpha_{p,i}+}$ are l-diverse. To facilitate the evaluation of these two additional conditions, for each action, the algorithm can precompute the maximal value and the summation of *weight* values of all its suffixes. Clearly, such computation can be an integrated part of reading inputs, and does not increase the order of computational complexity. Thus, the mvmdDiversity algorithm has $O(n_p \times n^2)$ time complexity in the worst case and $O(n_p \times n \times logn)$ in average cases (detailed algorithm descriptions are omitted due to space limitations).

VI. EVALUATION

In this section, we evaluate the effectiveness of our solutions and efficiency through experiments with real world Web applications. First, Section VI-A discusses the implementation of our techniques. Section VI-B then elaborates on the experimental settings. Finally, Section VI-C and VI-D present experimental results of the communication, computation, and processing overhead, respectively.

A. Implementation Overview

In previous sections, we have presented algorithms for determining the amount of padding for each flow given the vector-action set. To incorporate our techniques into an existing Web application requires following three steps. First, gather information about possible action-sequences and corresponding vector-sequences in the application. Second, feed the vector-action sets into our algorithms to calculate the desired amount of padding. Third, implement the padding according to the calculated sizes. The main difference between implementing an existing method (such as rounding) and ceiling padding lies in the second stage. Thus, we have focused on this stage in this paper. Nonetheless, we will also briefly describe how to collect the vector-action sets in Section VI-B and how to facilitate the third stage.

One may question the practicality of gathering information about possible action-sequences since the number of such sequences can be very large. However, we believe it is practical for most Web applications due to following three facts. First, the aforementioned side-channel attack on web applications typically arises due to highly interactive features, such as auto-suggestion. The very existence of such features implies that the application designer has already profiled the domain of possible inputs (that is, action-sequences) for implementing the feature. Therefore, such information must already exist in certain forms and can be easily extracted at a low cost. Second, even though a Web application may take infinite number of inputs, this does not necessarily mean there would be infinite action-sequences. For example, a search engine will no longer provide auto-suggestion feature once the query string exceeds a certain length. Third, all the three steps mentioned above could be part of the off-line processing, and would only need to be repeated when the Web application undergoes a redesign.

Note that implementing an existing padding method, such as rounding, will also need to go through the above three steps if

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

only padding cost is to be optimized. For example, without collecting and analyzing the vector-action sets, a rounding method cannot effectively select the optimal rounding parameter.

B. Experimental Setting

We collect testing vector-action sets from four real-world web applications, two popular search engines $engine^B$ and $engine^C$ (where users' searching keyword needs to be protected) and two authoritative information systems, $drug^B$ for drugs and $patent^C$ for patents, from two national institutes (where users' health information and company's patent interest need to be protected, respectively). Specially, for $engine^B$ and $engine^C$, we collect flow-vectors with respect to query suggestion widget for all possible four-letter combinations by crafting requests to simulate normal AJAX connection requests. For $drug^B$ and $patent^C$, we collect the vector-action set for all the drug and patent information by mouse-selecting following the applications' tree-hierarchical navigation. Such data can be collected by acting as a normal user of the applications without having to know their internal details. For our experiment, the data are collected using separate programs whose efficiency is not our main concern in this paper.

We observe that the flows of $drug^B$ and $patent^C$ are more diverse and larger than those of $engine^B$ and $engine^C$ evidenced by the standard deviations (σ) and the means (μ) of the flows, respectively. Besides, the flows of $drug^B$, $patent^C$ are much more disparate in values than those of $engine^B$, $engine^C$. Later we will show the effect of these different characteristics of flows on the costs.

All experiments are conducted on a PC with 2.20GHz Duo CPU and 4GB memory. We evaluate the overhead of computation, communication, and processing using execution time, padding cost ratio, and processing cost ratio, respectively. Specifically, for each application, we first obtain the total size of all flows ttl for all possible actions before padding, and then compute the padding cost $cost$ as shown in Definition 5 after padding. The padding cost ratio is formulated as $\frac{cost}{ttl}$. We also count the number of flows which need to be padded, and then formulate the processing cost ratio as the percent of flows to be padded among all flows. Clearly, given the interval Δ for random padding, theoretically the padding and processing cost ratio equal to $\frac{\Delta}{2\times ttl}$ and $1-\frac{1}{\Delta}$ respectively. Thus, we omit the comparison with it through the experiments.

To facilitate comparison, we use the $engine^B$ and $drug^B$ sets to compare the overheads for k-indistinguishability against an existing padding method, namely, packet-size rounding (simply rounding) [14], and the $engine^C$ and $patent^C$ sets to compare those for l-diversity against the other, namely, *maximizing* (a naive solution which pads each to be maximal size in the corresponding flow). For rounding, we set the rounding parameter $\Delta = 512$ and $\Delta = 5120$ for $engine^B$ and $drug^B$, respectively. Note that these Δ values just lead to results satisfying 5-indistinguishability in the padded data, and are adapted only for the comparison purpose. For l-diversity, we assign each action a uniformly random integer in a given range as its weight value (default $[1, 50]$). Note that our algorithms ensure the privacy for *l-diverse* vector-action sets and report exception for other sets regardless of the distribution and values of weights, and in real-life, the weight value could be assigned based on such as statistical results.

C. Communication Overhead

We first evaluate the communication overhead of our algorithms in the case of length-one action-sequences. In such cases, the svmdGreedy and svmdDiversity algorithms are equivalent to mvmdGreedy and mvmdDiversity, respectively. To apply the svsdSimple, svsdDiversity algorithms, we generate four vector-action sets by synthesizing the flow-vectors for the last action of the four collected sets.

For k-indistinguishability, figure 2(a) shows padding cost of each algorithm against k. Compared to rounding [14], our algorithms have less padding cost, while svmdGreedy incurs significantly less cost than that of rounding.

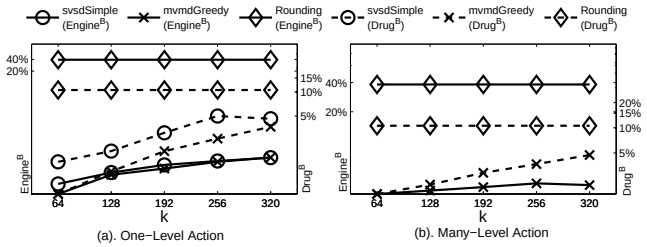


Fig. 2. Padding Cost Overhead Ratio (k-Indistinguishability)

For l-diversity, figure 3(a) shows padding cost of each algorithm against l. From the results, the padding costs of our algorithms are significantly less than that of *maximizing*. We observe that our algorithms are superior specially when the number of flow-vectors in a vector-action set is larger since our algorithms have high possibility to partition the flow-vectors with close value into padding group.

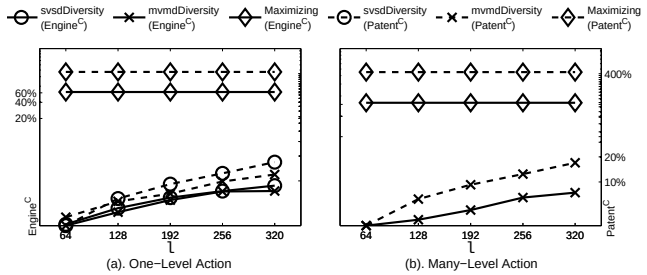


Fig. 3. Padding Cost Overhead Ratio (l-Diversity)

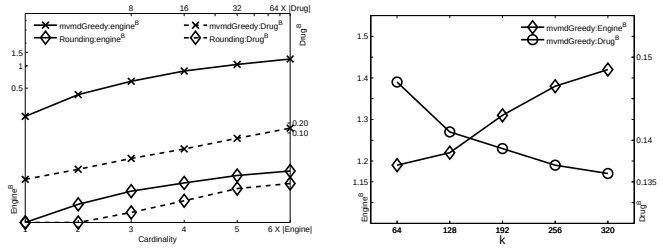
We then compare our algorithms with existing methods in the case of action-sequences of lengths larger than one. Figure 2(b) and 3(b) show padding costs of our mvmdGreedy and the rounding algorithm against k, and our mvmdDiversity and the maximizing algorithm against l, respectively. Rounding and maximizing incur larger padding cost than mvmdGreedy and mvmdDiversity in all cases. For example, the padding cost ratio of maximizing for $patent^C$ is prohibitively high as 418%, which is 140 times higher than that of mvmdDiversity when $l = 64$. The reason for mvmdGreedy, mvmdDiversity algorithms have more padding cost in the case of many-level action than in one-level is as follows. In many-level action, these algorithms first partition each vector-action set (except VA_1) into padding groups based on the prefix of actions and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

regardless of the values of flow-vectors. Besides, the further ordering by the *weight* in mvmdDiversity results in slightly more overhead than mvmdGreedy when $l = k$.

D. Computational Overhead

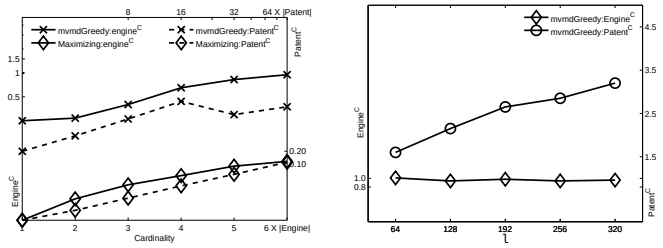
We first study the computation time of our mvmdGreedy and mvmdDiversity against the flow data cardinality n as shown in Figure 4(a) and 5(a). We generate n-sized flow data by synthesizing $\frac{n}{\sum_i(|VA_i|)}$ copies of the four collected vector-action sets. We set $k(l) = 160$ for this set of experiments, and conduct each experiment 1000 times and then take the average.



(a) Flow Data Cardinality (b) Privacy Property

Fig. 4. Execution Time in Seconds (k-Indistinguishability)

As the results show, our algorithms are practically efficient (1.2s and 0.98s for 2.7m flow-vectors for mvmdGreedy and mvmdDiversity, respectively) and the computation time increases slowly with n, although our algorithms require slightly more overhead than rounding (when it is applied to a single Δ value) and maximizing. However, this is partly at the expense of worse performance in terms of padding cost. Note that the slight reduction of execution time observed in Figure 5(a) for $patent^C$ at $32\times|patent^C|$ is reasonable since: first, the cardinality of each initial padding group based on the adjacent-prefixes may be smaller, which leads to less accumulated sorting time. Second, doubling the size of vector-action sets may result in less execution time based on the complexity in the average and worst cases ($2nlog2n < n^2$).



(a) Flow Data Cardinality (b) Privacy Property

Fig. 5. Execution Time in Seconds (l-Diversity)

We then study computation time against privacy property k on the two synthesized vector-action sets ($6\times engine^B$ and $64\times drug^B$), and against l on the other two sets ($6\times engine^C$ and $64\times patent^C$). As expected, rounding and maximizing are insensitive to k and l since they do not have the concept of k and l, respectively. On the other hand, a tighter upper bound on the time required for mvmdGreedy is $O(n_p\times n\times 2k\times\lambda)$ in the worse case and $O(n_p\times n\times log(2k\times\lambda))$ in the average case, where λ is the maximal number of actions which has the same prefix in all action-sequences. Clearly, when λ is $O(n)$, the computational complexity here is equivalent to that in Section IV-C.

The reason for this tighter upper bound is that mvmdGreedy always feeds a vector-action set with maximal $2k\times\lambda$ cardinality to svmdGreedy (except VA_1 whose size is 26, a constant, for $search^B$), since: first, for each vector-action set VA_i, mvmdGreedy first partitions it into padding groups based on the prefix (which has $O(|VA_i|)$ solution). Second, there are at most $2k$ adjacent-prefixes in same padding group of VA_{i-1}. Therefore, when $2k\times\lambda \ll n$, the execution time of mvmdGreedy should be in the range of $[log(2k\times\lambda), 2k\times\lambda]$ times of $O(n_p\times n)$ which is the execution time of rounding algorithm. These two datasets in our experimental environment satisfy above condition, for example, $26(\lambda)\times 320(k) \ll 2.7m$ for $search^B$. Observe that, mvmdDiversity does not satisfy the tighter upper bound since a vector-action set with size larger than $2l$ probably cannot be split into two l-diverse subsets.

Figure 4(b) illustrates the computation time of mvmdGreedy against the privacy property k. Interestingly, the computation time increases slowly (from $1.19s$ to $1.42s$) with k for $engine^B$, and decreases slowly (from $0.147s$ to $0.136s$) for $drug^B$. Stress that the results are reasonable since both results fall within the expected range. Figure 5(b) shows that the computational time of mvmdDiversity increases slowly with l for $patent^C$, and is almost same for different l in the case of $engine^C$.

Costs may also be incurred for actually implementing the padding. Thus, we must also minimize the number of packets to be padded. For this purpose, we evaluate the processing cost ratio, which captures the proportion of flow-vectors to be padded among all such vectors.

Figure 6 shows the processing cost of each algorithm against k. Rounding algorithm must pad each flow-vector regardless of the k's and the applications, while our algorithms have much less cost for $engine^B$ and slightly less for $drug^B$.

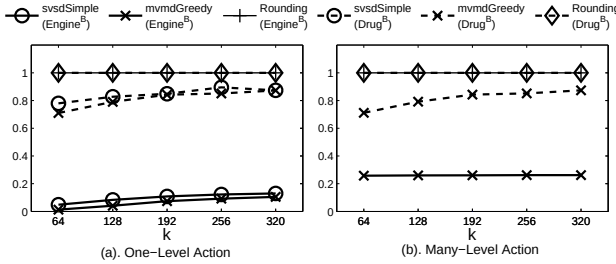


Fig. 6. Processing Cost Overhead Ratio (k-Indistinguishability)

Similarly, from the results of the processing cost against l shown in Figure 7, we can see that maximizing algorithm almost pads each flow-vector regardless of the l's and the applications, while our algorithms have much less cost for $engine^C$ and slightly less for $patent^C$.

VII. Related Work

In this section, we briefly review existing efforts on side channel attacks and privacy preserving in Web applications.

Side-Channel Attack: Various side-channel leakages have been extensively studied in the literature. By measuring the amount of time taken to respond to the queries, an attacker

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2014.2302308, IEEE Transactions on Dependable and Secure Computing

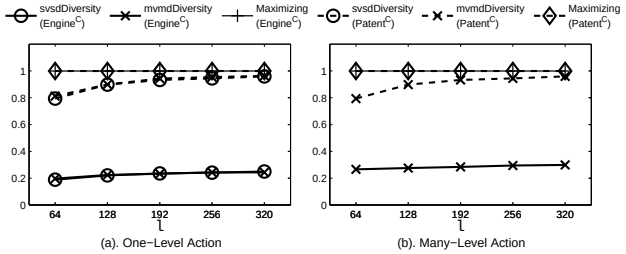


Fig. 7. Processing Cost Overhead Ratio (l-Diversity)

may extract OpenSSL RSA privacy keys [10], and similar timing attacks are proved to be still practical recently [9]. By differentiating the sounds produced by keys, an attacker with the help of the large-length training samples may recognize the key pressed [3]; Zhuang *et al.* further present an alternative approach to achieving such attack which does not need the training samples [43]. By exploiting queuing side channel in routers by sending probes from a far-off vantage point, an attacker may fingerprint websites remotely against home broadband users [22], [21]. Ristenpart *et al.* discover cross-VM information leakage on Amazon EC2 based on the sharing of physical infrastructure among users [33]. Search histories may be reconstructed by session hijacking attack [12], while web-browsing histories may be compromised by cache-based timing attacks [18]. Saponas *et al.* show the transmission characteristics of encrypted video streaming may allow attackers to recognize the title of movies [35].

Meanwhile, much efforts have been made on developing techniques to mitigate the threats of such leakages. Countermeasures based on traffic-shaping mechanisms (such as, padding, mimicking, morphing, and so on) are suggested against the exposure of identification of encrypted web traffic in [37]. HTTPOS, a browser-side system, is proposed to prevent information leakages of encrypted HTTP traffic through configurable traffic transformation techniques in [29]. Timing mitigator is introduced to achieve any given bound on timing channel leakage by delaying output events to limit the amount of information in [2]. Zhang *et al.* present an approach to verifying the VMs' exclusive use of a physical machine. The approach exploits a side-channel in the L2 memory cache as a defensive detection tool rather than a vector of attack [42]. Provider-enforced deterministic execution by eliminating all the internal timing channels has been proposed to combat timing channel attack in cloud [4].

Privacy Preservation in Web Applications: The privacy preserving issue has received significant attentions in various domains, such as, data publishing and data mining [15] [34], network [5] [7], network [16] [32] [19], outsourced data [11] [39], multiparty computation [31], web applications [8] [14] [36], and so on.

In the context of web applications, many side-channel leakages in encrypted web traffic have been identified in the literature which allow to profile the web applications themselves and their internal states [14], [4], [22], [12]. Meanwhile, several approaches have been proposed to analyze and mitigate such leakages, such as [37], [29], [40], [2]. Recently, a black-box approach has been proposed to detect and quantify the side-channel vulnerabilities in web application by extensively crawling a targeted application [13]. Most recently, a formal framework is proposed to measure security in terms of the amount of information leaked from the observations without the assumption of any particular attacks [6].

The main inspiration of our work is the recent work by Chen *et al.* in [14] which demonstrated through case studies that side-channel problems are pervasive and exacerbated in web applications due to their fundamental features. Then the authors further study approaches to identifying such threats and quantifying the amount of information disclosed in [41]. They show that an application-agnostic approach generally suffers from high overhead and low level of privacy protection, and consequently effective solutions to such threats likely will rely on the in-depth understanding of the applications themselves. Finally, they design a complete development process as a fundamental solution to such side channel attacks. Our model and solutions provide finer control over the trade-off between privacy protection and cost, and those solutions can certainly be integrated into the development process.

Closer to our work, traffic morphing is proposed to mitigate the threats by traffic analyzing on packet sizes and sequences through network [40]. Although their proposed system morphs classes of traffic to be indistinguishable, traffic morphing pads or splits packets on the fly which may degrade application's performance. Further, due to the lack of privacy requirement, the degree of privacy, which the traffic transformation is able to achieve, cannot be evaluated during the process of padding, consequently, it cannot ensure the privacy being satisfied. In contrast, our proposed algorithms following the proposed model theoretically guarantee the desired privacy property.

The k-indistinguishability privacy requirement of this paper has been previously discussed in our previous works [27] [28], and we significantly extend the model to adapt l-diversity, and correspondingly formulate the problems, design the algorithms, and experimentally confirm the efficiency and effectiveness of the algorithms.

VIII. Conclusion

As Web-based applications become more popular, their security issues will also attract more attention. In this paper, we have demonstrated an interesting connection between the traffic padding issue of Web applications and the privacy-preserving data publishing. Based on this connection, we have proposed a formal model for quantifying the amount of privacy protection provided by traffic padding solutions. We have also designed algorithms by following the proposed model. Our experiments with real-world applications have confirmed the performance of our solutions to be superior to existing ones in terms of communication and computation overhead.

Acknowledgment

The authors thank the anonymous reviewers for their valuable comments. Authors with Concordia University are partially supported by Natural Sciences and Engineering Research Council of Canada under Discovery Grant N01035, Strategic Project Grant STPGP/396651-2010, and Canada Graduate Scholarship. Kui's research is supported in part by US NSF

through grant CNS-1262277 and CNS-1262275. Shunzhi's research is supported in part by the National Natural Science Foundation of China under Grants No. 61373147.

REFERENCES

[1] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, D. Thomas, and A. Zhu. Anonymizing tables. In *ICDT '05*, pages 246–258, 2005.

[2] A. Askarov, D. Zhang, and A.C. Myers. Predictive black-box mitigation of timing channels. In *CCS*, pages 297–307, 2010.

[3] D. Asonov and R. Agrawal. Keyboard acoustic emanations. *Security and Privacy, IEEE Symposium on*, page 3, 2004.

[4] A. Aviram, S. Hu, B. Ford, and R. Gummadi. Determinating timing channels in compute clouds. In *CCSW '10*, pages 103–108, 2010.

[5] M. Backes, G. Doychev, M. Dürmuth, and B. Köpf. Speaker recognition in encrypted voice streams. In *ESORICS '10*, pages 508–523, 2010.

[6] Michael Backes, Goran Doychev, and Boris Köpf. Preventing Side-Channel Leaks in Web Traffic: A Formal Approach. In *NDSS'13*, 2013.

[7] K. Bauer, D. Mccoy, B. Greenstein, D. Grunwald, and D. Sicker. Physical layer attacks on unlinkability in wireless lans. In *PETS '09*, pages 108–127, 2009.

[8] I. Bilogrevic, M. Jadliwala, K. Kalkan, J.-P. Hubaux, and I. Aad. Privacy in mobile computing for location-sharing-based services. In *PETS*, pages 77–96, 2011.

[9] BillyBob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In *ESORICS'11*, pages 355–371. 2011.

[10] D. Brumley and D. Boneh. Remote timing attacks are practical. In *USENIX*, 2003.

[11] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou. Privacy-preserving query over encrypted graph-structured data in cloud computing. In *ICDCS'11*, pages 393–402, 2011.

[12] C. Castelluccia, E. De Cristofaro, and D. Perito. Private information disclosure from web searches. In *PETS'10*, pages 38–55, 2010.

[13] Peter Chapman and David Evans. Automated black-box detection of side-channel vulnerabilities in web applications. In *CCS'11*, pages 263–274, 2011.

[14] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *IEEE Symposium on Security and Privacy'10*, pages 191–206, 2010.

[15] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k-anonymous data mining: A survey. In *Privacy-Preserving Data Mining: Models and Algorithms*. 2008.

[16] G. Danezis, T. Aura, S. Chen, and E. Kiciman. How to share your favourite search results while preserving privacy and quality. In *PETS'10*, pages 273–290, 2010.

[17] C. Dwork. Differential privacy. In *ICALP (2)*, pages 1–12, 2006.

[18] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *CCS '00*, pages 25–32, 2000.

[19] P. W. L. Fong, M. Anwar, and Z. Zhao. A privacy preservation model for facebook-style social network systems. In *ESORICS '09*, pages 303–320, 2009.

[20] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42:14:1–14:53, June 2010.

[21] X. Gong, N. Borisov, N. Kiyavash, and N. Schear. Website detection using remote traffic analysis. In *PETS'12*, pages 58–78. 2012.

[22] X. Gong, N. Kiyavash, and N. Borisov. Fingerprinting websites using remote traffic analysis. In *CCS*, pages 684–686, 2010.

[23] V. Kann. Maximum bounded h-matching is max snp-complete. *Inf. Process. Lett.*, 49:309–318, March 1994.

[24] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:881–892, July 2002.

[25] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient fulldomain k-anonymity. In *SIGMOD*, pages 49–60, 2005.

[26] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE '07*, pages 106–115, 2007.

[27] W. M. Liu, L. Wang, P. Cheng, and M. Debbabi. Privacy-preserving traffic padding in web-based applications. In *WPES '11*, pages 131–136, 2011.

[28] W. M. Liu, L. Wang, K. Ren, P. Cheng, and M. Debbabi. k-indistinguishable traffic padding in web applications. In *PETS'12*, pages 79–99, 2012.

[29] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS '11*.

[30] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.

[31] S. Nagaraja, V. Jalaparti, M. Caesar, and N. Borisov. P3ca: private anomaly detection across isp networks. In *PETS'11*, pages 38–56, 2011.

[32] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy '09*, pages 173–187, 2009.

[33] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *CCS*, pages 199–212, 2009.

[34] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. on Knowl. and Data Eng.*, 13(6):1010–1027, 2001.

[35] T. S. Saponas and S. Agarwal. Devices that tell on you: Privacy trends in consumer ubiquitous computing. In *USENIX '07*, pages 5:1–5:16, 2007.

[36] J. Sun, X. Zhu, C. Zhang, and Y. Fang. Hcpp: Cryptography based secure ehr system for patient privacy and emergency healthcare. In *ICDCS'11*, pages 373–382, 2011.

[37] Q. Sun, D. R. Simon, Y. M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy '02*, pages 19–, 2002.

[38] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[39] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *ICDCS'10*, pages 253–262, 2010.

[40] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS '09*.

[41] K. Zhang, Z. Li, R. Wang, X. Wang, and S. Chen. Sidebuster: automated detection and quantification of side-channel leaks in web application development. In *CCS '10*, pages 595–606, 2010.

[42] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, pages 313–328, 2011.

[43] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Trans. Inf. Syst. Secur.*, 13(1):3:1–3:26, November 2009.

Wen Ming Liu is currently a PhD student with Computer Science at Concordia University, Canada with CGS scholarship from NSERC. He received his M.Sc. degrees from Fuzhou University, China in 1997, and Concordia University, Canada in 2008. His research interests include data privacy, cryptology, and network security.

Lingyu Wang is an associate professor in the Concordia Institute for Information Systems Engineering (CIISE) at Concordia University, Montreal, Quebec, Canada. He received his Ph.D. degree in Information Technology from George Mason University. He holds a M.E. from Shanghai Jiao Tong University and a B.E. from Shenyang Aerospace University in China. His research interests include data privacy, network security, and security metrics. He has co-authored over 80 refereed conference and journal articles.

Pengsu Cheng is a security analyst at Gameloft, working on vulnerability analysis. He received his M.A.Sc. degree in Information Systems Security from Concordia University, Canada in 2011, and B.E. in Electronic Information Engineering from Wuhan University, China in 2009. His research interests include security metrics, vulnerability analysis and intrusion detection.

Kui Ren is an associate professor of CSE Dept at SUNY Buffalo. He received his PhD degree from Worcester Polytechnic Institute. Kui's research interests include Cloud Security, Wireless Security, and Smartphone-enabled Crowdsourcing Systems. His research has been supported by NSF, DoE, AFRL, and Amazon. He is a Senior Member of IEEE and a member of ACM.

Shunzhi Zhu is a Professor of the School of Computer and Information Engineering at Xiamen University of Technology, China. He received his Ph.D. degree in Control Theory and Control Engineering from Xiamen University, China. His current research interests include data mining, system engineering, database security.

Mourad Debbabi received the Ph.D. and M.Sc. degrees in computer science from Paris-XI Orsay, University, France. He is currently a Full Professor at the Concordia University, Montreal, Quebec, Canada. He holds the Concordia Research Chair Tier I in Information Systems Security. He is also the President of the National Cyber Forensics Training Alliance (NCFTA Canada). He published more than 170 research papers in journals and conferences on computer security, digital forensics, Java security, cryptographic protocols, malicious code detection and network security.