k-Indistinguishable Traffic Padding in Web Applications

Wen Ming Liu[1], Lingyu Wang[1], Kui Ren[2], Pengsu Cheng[1], and Mourad Debbabi[1]

[1] Concordia Institute for Information Systems Engineering, Concordia University
[2] Department of Electrical and Computer Engineering, Illinois Institute of Technology

Abstract. While web-based applications are becoming increasingly ubiquitous, they also present new security and privacy challenges. In particular, recent research revealed that many high profile Web applications might cause private user information to leak from encrypted traffic due to side-channel attacks exploiting packet sizes and timing. Moreover, existing solutions, such as random padding and packet-size rounding, are shown to incur prohibitive cost while still not ensuring sufficient privacy protection. In this paper, we propose a novel k-indistinguishable traffic padding technique to achieve the optimal tradeoff between privacy protection and communication and computational cost. Specifically, we first present a formal model of the privacy-preserving traffic padding (PPTP). 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.

1 Introduction

Web-based applications are gaining popularity. By providing software services through Web browsers, such applications demand less client-side resources and are easier to deliver and maintain than their desktop counterparts. However, they also present new security and privacy challenges partly because the untrusted Internet now becomes an integral part of the application for carrying the continuous interaction between users and service providers. 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 [9]. By analyzing packets' sizes and timing, an eavesdropper can potentially identify an application's internal state transitions as well as users' inputs. Moreover, such side-channel attacks are shown to be pervasive and fundamental to Web applications due to their intrinsic characteristics , such as low entropy inputs, rich and diverse resource objects, and stateful communications.

For example, Table 1 shows the size and direction of packets observed between users and a popular real-world search engine. Observe that due to the auto-suggestion feature, with each keystroke, the browser sends a b-byte packet to the server; the server then replies with two packets of 54 bytes and s bytes, respectively; finally, the browser sends a 60-byte packet to the server. In addition, in the same input string, each subsequent keystroke increases the b value by one byte, and the s value depends not only on the current keystroke but also on all the previous ones. Clearly, an eavesdropper can pinpoint packets corresponding to an input string from observed traffic by the packets with fixed pattern in size(first, second, and last), even though the traffic has been encrypted. In this paper, we assume such a worst case scenario in which an eavesdropper can identify traffic related to a Web application (such as using de-anonymizing techniques [27]) and locate packets for user inputs using the above technique.

User Input	Observed Directional Packet Sizes
`a`	$b_1 \rightarrow, \quad \leftarrow 54, \ \leftarrow 509, \quad 60 \rightarrow$
`00`	$b_2 \rightarrow, \quad \leftarrow 54, \ \leftarrow 505, \quad 60 \rightarrow,$
	$b_2 + 1 \rightarrow, \leftarrow 54, \ \leftarrow 507, \quad 60 \rightarrow$
	(b bytes) (s bytes)

Table 1. User Inputs and Corresponding Packet Sizes

Moreover, the size of the third packet(s) will provide a good indicator of the input itself. Specifically, the left tabular of Table 2 shows the s value for each character entered as the first keystroke of an input string. We can see that six characters can be uniquely identified with this s value. The right tabular shows the s value for a character entered as the second keystroke. In this case, the s value for each character in the right tabular is different from that in the left, 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 about the two consecutive keystrokes shown in both tables (for simplicity, we are only considering four characters here, whereas in reality it may take more than two keystrokes to uniquely identify an input string).

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**
509	504	502	516	499	504	502	509	492
j	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**
517	499	501	503	488	509	525	494	498
s	**t**	**u**	**v**	**w**	**x**	**y**	**z**	
488	494	503	522	516	491	502	501	

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

Table 2. s Value for Each Char Entered as the First or Second Keystroke (Left or Right Tabular)

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. However, such a solution does not come free, since padding packets will result in additional overhead. In fact, it has been shown that a straightforward solution, such as random padding and rounding, may incur a prohibitive overhead (e.g. 21074% for a well-known online tax system [9]). Moreover, such an application-agnostic approach typically aim to maximize, but cannot guarantee, the amount of privacy protection.

In Table 3, we consider a different way for padding the packets. The first and last columns respectively show the s value and corresponding input (the second keystroke). 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 characters 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 their corresponding s values are all identical to the maximum value. Thus the characters inside each padding group will no longer be distinguishable from each other based on their s values. The objective now is to find a padding option that can provide sufficient privacy protection and meanwhile minimize the padding cost.

s **Value**	**Padding**		(1^{st} **Keystroke**) 2^{nd} **Keystroke**
	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 3. Mapping PPTP to PPDP

Interestingly, this *privacy-preserving traffic padding (PPTP)* problem can be naturally interpreted as another well studied problem, *privacy-preserving data publishing (PPDP)* [13]. To revisit Table 3, if we regard the s value as a *quasi-identifier* (such as DoB), the input as a *sensitive value* (such as medical condition), and the padding options as different ways for generalizing the DoB into *anonymized groups* (for example, by removing the day from a DoB), then we immediately have a classic PPDP problem, that is, publishing DoBs and medical conditions while preventing adversaries from linking any published medical condition to a person through his/her DoB.

The similarity between the two problems implies we may borrow many existing efforts in the PPDP domain to address the PPTP issue. On the other hand, there also exist significant differences between them. For example, in Table 3, the second option will typically be considered as worse (than the first) in PPDP since it results in larger anonymized groups, whereas it is actually better in terms of padding cost (totally 24 bytes, in contrast to 33 by the first option). As another example, we will show later that the effect of combining two keystrokes will be equivalent to releasing multiple inter-dependent tables, which actually leads to a novel PPDP problem.

In this paper, we first briefly review the formal model of the PPTP issue based on the mapping to PPDP which introduced in our short version [18]. 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 padding cost. Finally, we evaluate the effectiveness and efficiency of our algorithms by comparing them to existing solutions through experiments with real-world Web applications.

The contribution of this paper is threefold. First, the identified similarity between PPTP and PPDP establishes a bridge between the two research communities, 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 PPTP model is among the first efforts on formally addressing this issue (in contrast to our work, the formal model given by Chen *et al.* [9] lacks a clear definition of privacy requirements and only considers two application-agnostic padding methods). Third, the proposed padding algorithms can lead to practical solutions for real world Web applications, as evidenced by our experiments.

The rest of the paper is organized as follows. Section 2 defines our PPTP model. Section 3 formulates PPTP problems and analyzes the complexity. Section 4 devises heuristic algorithms for the formulated problems. Section 5 experimentally evaluates the performance of our algorithms. Section 6 discusses the extensions and implementation of our solution. Section 7 reviews related work and Section 8 concludes the paper.

2 The Model

To be self contained, we briefly repeat here the PPTP model introduced in our short version [18], and shall delay the discussion about extending it to encompass l-diversity in Section 6. Table 4 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 4. The Notation Table

2.1 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 2 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 related 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. Actions inside an action-sequence are separated into different action-sets since their relationship is known from traffic patterns and thus padding them together will not work (preventing such inferences about the application's state transitions comprises a future direction).

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 "a" and "0**0**" in Table 1 to be the only possible inputs, there are two action-sequences a and 00, and two action-sets $A_1 = \{a, 0\}$ and $A_2 = \{\mathbf{0}\}$. $\boxdot$

Definition 2 models concepts related to the observation made by an eavesdropper. Note that a *flow-vector* is intended to only model those packets that may contribute to identify an action (such as the s value in Table 1). Further, a vector-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-vector v *as a sequence of* flows *where each flow* s *is an integer (a directional packet size). An action corresponds to a flow-vector based on packets it triggers.*

- *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 three flow-vectors, $v_1 = 509$, $v_2 = 505$, and $v_3 = 507$ (note that we only model those packets whose sizes can help to identify an action), corresponding to actions a, 0 (as first keystroke), and 0 (as second keystroke), respectively. We have two vector-sequences, v_1 and v_2v_3, corresponding to action-sequences a and 00, respectively. We also have two vector-sets $V_1 = \{509, 505\}$ and $V_2 = \{507\}$ corresponding to the two action-sets A_1 and A_2 in Example 1. $\boxdot$

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* $\{(v, a) : v \in V_i \wedge a \in A_i\}$.

Example 3. Following above examples, given the action-set A_1 and vector-set V_1, then the vector-action set is $VA_1 = \{(509, a), (505, 0)\}$. Similarly, $VA_2 = \{(507, \mathbf{0})\}$. $\boxdot$

2.2 Privacy and Cost Model

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. In this case, we can map a given vector-action set $VA = \{(v, a) : v \in V \wedge a \in A\}$ 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.

Inspired by k-anonymity [24] in PPDP domain, 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. Second, in theory k could always be set to be sufficient large to provide enough confidentiality, although we believe 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 possible inputs, we believe focusing on protecting sensitive user input (by hiding it among other possible inputs) yields higher practical feasibility and significance than on perfect confidentiality (attempting to hide everything).

Furthermore, such mapped PPDP problems actually possess a unique characteristic. That is, the sensitive values (actions) are always unique. Thus, by satisfying k-indistinguishability, the vector-action set also satisfies *l-diversity* ($l = k$) in its simplest form [20]. We will also apply more general forms of l-diversity to address cases where not all actions should be treated equally in padding, as sketched in Section 6. Furthermore, a probabilistic approach based on *differential privacy* [12] is another possible extension to enhance our model such that the padding result will be immune to eavesdroppers' prior knowledge. Nonetheless, this simple model is sufficient to demonstrate the usefulness of mapping PPTP to PPDP.

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.*

2.3 The SVMD and MVMD Cases

In the previous section, we 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 flows 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). *We define*

- *the* i-prefix *of an action-sequence $\vec{a} = (a_1, a_2, \ldots, a_t)$ ($i \in [1, t]$), denoted as $pre(\vec{a}, i)$, as the sequence $(a_1, a_2, \ldots, a_i)$, and we say a_{i-1} is the* adjacent-prefix *(or simply prefix) of a_i.*
- *similarly, we define the* i-prefix *of vector-sequence $\vec{v}$, and the* adjacent-prefix *of 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.

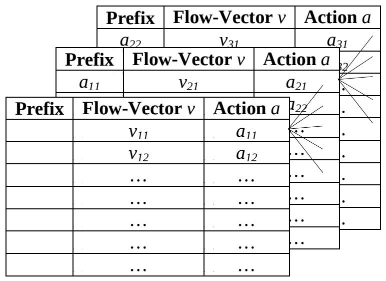


Fig. 1. The Vector-Action Set in MVMD Case

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

3 PPTP Problems

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 3.1, and then address the SVSD and SVMD cases in Section 3.2 and the MVMD case in Section 3.3.

3.1 Padding Method

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 [9]. 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 shown in the left tabular of Table 2 to a multiple of 128 (for example, 509 to $4 \times 128 = 512$). It can be shown that such rounding can achieve 5-indistinguishability (detailed calculations will be omitted due to space limitations). However, increasing the rounding size to 512 can still only satisfy 5-indistinguishability, whereas further increasing it to 520 will actually only satisfy 2-indistinguishability.

On the other hand, as demonstrated in Section 1, we can now apply the PPDP technique of generalization to addressing the PPTP problem. A generalization technique will partition the vector-action set into padding 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 it, or replace it with a range of values like in normal generalization. The above considerations lead to a new padding method given in Definition 9. Basically, after partitioning a vector-action set into padding groups, we pad each flow in a padding group to be identical to 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.*

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.

3.2 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 $P^V = \{P_1, P_2, \dots, P_m\}$, 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. $\boxdot$

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 $k \leq\mid V \mid$, find a partition P^{VA} on VA such that the corresponding partition on V, denoted as $P^V = \{P_1, P_2, \dots, P_m\}$, 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. $\boxdot$

Theorem 1 shows that the above PPTP problem is intractable. The proof is omitted due to space limitations (basically, we prove the result through a reduction to the problem of *Edge Partition Into Triangles* (EPIT) [14]). Theorem 1 indicates that Problem 2 is NP-hard even when there are only two different flow values (that is, the sizes of packets in the traffic) in the vector-set.

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

Note that, at first glance, the SVMD problem may resemble the problem of *k-means clustering* [15]. 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.

3.3 MVMD Problem

As mentioned in Section 2.3, 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 a toy example of auto-suggestion feature.

Example 4. In Table 5, the second column shows each flow corresponding to c_1, c_2, c_3, c_4 when entered as the first keystroke, respectively. Similarly, the 16 cells c_{ij} in row: $(i \in [2-5])$ and column: $(j \in [3-6])$ show the flow corresponding to the second keystroke when c_{i-1} is the first keystroke and c_{j-2} the second keystroke.

Suppose an eavesdropper has observed the flow for the second keystroke. In order to preserve 2-indistinguishability with minimal padding overhead, we will partition the 16 cells into eight groups $P_i = \{c_{jk} : \frac{c_{jk}}{10} = i\}$, such that the size of each group is not less than 2. For example, the queried strings c_1c_1 and c_3c_2 are in one group and c_1c_1 should be padded to 15. When the eavesdropper observes that the flow for the second keystroke is 15, she cannot determine whether the queried string is c_1c_1 or c_3c_2.

However, suppose that the eavesdropper also observes the flow corresponding to the first keystroke, they can determine that the first keystroke is either c_1 or c_3 when the flow is 5 or 15, respectively. Consequentially, the eavesdropper can eventually infer the queried string by combining these observations. $\boxdot$

		c_1	c_2	c_3	c_4
c_1	5	10	20	80	50
c_2	10	40	70	30	60
c_3	15	65	15	45	75
c_4	20	35	55	85	25

Table 5. Padding in the MVMD Case

One seemingly valid solution is padding the flow-vector for each keystroke so that 2-indistinguishability is satisfied separately for each keystroke. Unfortunately, this will fail to satisfy 2-indistinguishability. To pad traffic for the first keystroke, the optimal solution is to partition $\{5, 10, 15, 20\}$ into two padding groups, $\{5, 10\}$ and $\{15, 20\}$. However, when the eavesdropper observes the flow corresponding to the first keystroke, he/she can still determine it must be either c_1 or c_3 when the size is 10 or 20, respectively, because only when the first keystroke starts with c_1 or c_3 can the flow for second keystroke be padded to 15. Therefore, the eavesdropper will eliminate c_2 and c_4 from possible guesses, which violates 2-indistinguishability.

Another seemingly viable solution is to first collect all vector-sequences for the sequence of keystrokes and then pad them such that the current input string as a whole cannot be distinguished from at least $k-1$ others. Unfortunately, such an approach cannot guarantee the privacy property, either. First, the auto-suggestion feature requires the server to immediately respond to the client upon each single keystroke. Second, when receiving a single keystroke, the server cannot predict what would be the next input and hence cannot decide which padding option is suitable. For example, suppose the flow corresponding to c_1 in c_1c_2 should be padded to 10, while in c_1c_3 to 15. When the server receives c_1, it cannot determine whether to pad c_1 to 10 or to 15.

The above 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 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 aforementioned conditions, no matter how an eavesdropper analyzes traffic information, either for an action alone or combining multiple observations of previous actions, the mental image about the actual action-sequence (or any of its subsets) remains the same (detailed proof is omitted due to space limitations). 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 $P^{V_i} = \{P^i_1, P^i_2, \ldots, P^i_{m_i}\}$ on V_i satisfies

- $\forall((i \in [1, t-1]) \wedge (j \in [1, m_i])) \begin{cases} |P^i_j| \geq k, & \text{if } (|V_i| \geq k), \\ |P^i_1| = |V_i|, & \text{if } (|V_i| < k); \end{cases}$
- $\forall(j \in [1, m_t]), |P^t_j| \geq k;$
- 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. $\boxdot$

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

4 The Algorithms

In this section, we design three algorithms for partitioning the vector-action set into padding groups to satisfy a given privacy requirement. Our intention is not to design an exhaustive list of solutions but rather to demonstrate the existence of abundant possibilities in approaching this PPTP issue.

4.1 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 next section). The svsdSimple algorithm shown in Table 6 basically attempts to minimize the cardinality of padding groups in the SVSD case. 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.

```
Algorithm svsdSimple
Input: a vector-action set $VA$, the privacy property $k$;
Output: the partition $P^{VA}$ of $VA$;
Method:
1.  Let $P^{VA} = \phi$;
2.  Let $S^{VA}$ be the sequence of $VA$ in a non-decreasing order of $V$;
3.  Let $N = \frac{|S^{VA}|}{k}$;
4.  For $i = 0$ to $N - 2$
5.      Let $P_i = \bigcup_{j=i\times k}^{(i+1)\times k-1}(S^{VA}[j])$;
6.      Create partition $P_i$ on $P^{VA}$;
7.  Create $P_{N-1} = \bigcup_{j=(N-1)\times k}^{|S^{VA}|-1}(S^{VA}[j])$ on $P^{VA}$;
8.  Return $P^{VA}$;
```

Table 6. The svsdSimple Algorithm for SVSD-Problem

More specifically, svsdSimple first sorts each single flow in the flow-vector into a non-decreasing order of the flows, and then selects k pairs of (flow-vector, action) each time in that order to form a padding group. This is repeated until the number of pairs is less than k. The remainder of pairs is simply appended to the last padding group.

The computational complexity is $O(nlogn)$ where $n = |VA|$, since step 2 costs $O(nlogn)$ time and each (flow-vector, action) pair is considered once for the remaining steps.

4.2 The svmdGreedy Algorithm

The svmdGreedy algorithm, which aims at both SVSD and SVMD problems, is shown in Table 7. 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 $(P_{i1} \cup P_{i2} = P_i) \wedge (P_{i1} \cap P_{i2} = \phi) \wedge (|P_{i1}| \geq k) \wedge (|P_{i2}| \geq k)$. 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 of flow-vectors 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 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 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 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 7. 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 $n = |VA|$.

4.3 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 3.3 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 8. 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 algorithm 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 8. The mvmdGreedy Algorithm For MVMD-Problem

5 Evaluation

In this section, we evaluate the effectiveness of our solutions and efficiency through experiments with real world Web applications. Section 5.1 first elaborates on the experimental settings. Then, Section 5.2, 5.3, and 5.4 present experimental results of the communication, computation, and processing overhead, respectively.

5.1 Experimental Setting

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

Information about data cardinality and action levels is shown in Table 9(a), and information about the distribution and distinct number of data sizes is shown in Table 9(b). We observe that the flows of $drug^B$ are more diverse than those of $engine^B$ evidenced by the standard deviations (σ) of the flows. The flows of $drug^B$ are also larger than those of $engine^B$ based on their means (μ). Besides, the flows of $drug^B$ are much more disparate in values than those of $engine^B$. For example, there are only 889 different flow sizes among 456976 flow-vectors in $engine^B$, while there are 1015 different among 4883 vectors in $drug^B$. Later we will show the effect of these different characteristics of flows on the costs.

	$engine^B$	$drug^B$
Level Number	4	3
1	26,	1,
2	676,	27,
3	17.6K,	4883
4	457K	-
Vector Number in Total	475254	5091

(a).The Number of Vectors

Level	$engine^B$			$drug^B$		
	μ	σ	#	μ	σ	#
1	936	71	23	-	0	1
2	896	70	229	37833	22102	27
3	768	216	741	23411	9796	1015
4	429	173	889	-	-	-

(b). Distribution of Data Sizes

Table 9. Flow Data Outline of $engine^B$ and $drug^B$

Note that the size information collected through our programs may have integrally shifted from the original one. However, we argue that such information is still sufficient and reasonable for our experimental evaluation due to following facts. First, the collected data preserve adequate characteristics of the original data with respect to the traffic-size distinction. Second, although the length of HTTP request and response may vary due to different browsers and platforms, the variance is constant for the same setting and can be determined in advance. Also, the size information may vary when adopting compression in the web objects or HTTP body. Our solutions regard such variances as different inputs.

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 of traffic padding 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.

5.2 Communication Overhead

We first compare the communication overhead of our algorithms against an existing padding method, namely, packet-size rounding (simply rounding) [9]. 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. The first set of experiments evaluate svsdSimple, svmdGreedy, and mvmdGreedy algorithms. To apply the svsdSimple algorithm, we generate two vector-action sets by synthesizing the flow-vectors for the last action of $engine^B$, $drug^B$, respectively. Note that the svmdGreedy and mvmdGreedy algorithms are equivalent with length-one action sequences.

Figure 2(a) shows padding cost of each algorithm against k. Compared to rounding [9], our algorithms have less padding cost, while svmdGreedy incurs significantly less cost than that of rounding. Table 10(a) shows the details of padding cost overhead ratio in percentage for $k = 192$. 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.

We then compare the mvmdGreedy with rounding algorithm in the case of action-sequences of lengths larger than one. Figure 2(b) shows padding cost of our mvmdGreedy algorithm and rounding algorithm against k. Packet-size rounding incurs larger

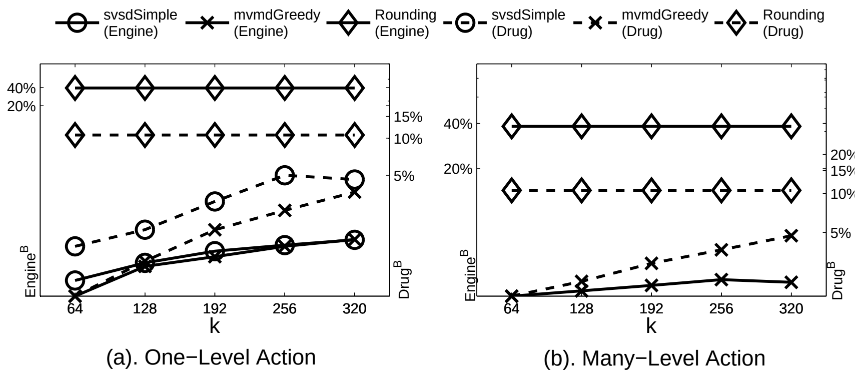


Fig. 2. Padding Cost Overhead Ratio

Application	svsdSimple	mvmdGreedy	Rounding
$Engine^B$	0.0748	0.0604	39.4043
$Drug^B$	3.0743	1.8097	10.5922

(a). One-level Action

Application	mvmdGreedy	Rounding
$Engine^B$	3.3297	38.2659
$Drug^B$	2.8864	10.5672

(b). Many-level Action

Table 10. Padding Cost Overhead Ratio When $k = 192$

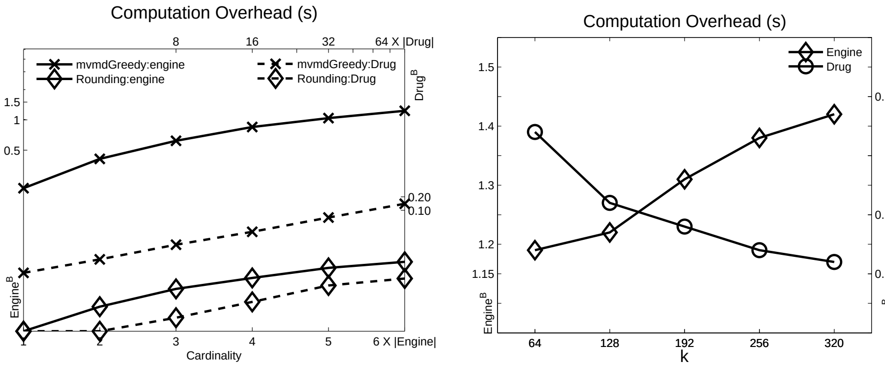
padding cost than mvmdGreedy in all cases. Note that, rounding will get worse when Δ is set to be larger to minimize an eavesdropper's capability of inference [9]. For example, the padding ratio is 118% for $drug^B$ when applying rounding to make all drug information indistinguishable. Table 10(b) shows the detail of overhead ratio in percentage for $k = 192$. The reason for mvmdGreedy algorithm has more padding cost in the case of many-level action than in one-level is as follows. In many-level action, mvmdGreedy first partition each vector-action set (except VA_1) into padding groups based on the prefix of actions and regardless of the values of flow-vectors.

5.3 Computational Overhead

Figure 3(a) illustrates the computation time of mvmdGreedy algorithm and rounding algorithm against the flow data cardinality n. We generate n-sized flow data by synthesizing $\frac{n}{\sum_i(|VA_i|)}$ copies of $engine^B$, $drug^B$ respectively. We set $k = 160$ for this set of experiments, and conduct each experiment 1000 times and then take the average.

As the results show, the mvmdGreedy algorithm is practically efficient (1.2s for 2.7m flow-vectors) and the computation time increases slowly with n, although our algorithm requires slightly more overhead than rounding when it is applied to a single Δ value. However, this is partly due to the application-agnostic nature of the rounding method, which results in worse performance in terms of padding cost. Also, as shown in the previous section, such a method may require to test many Δ values for an optimal choice since larger values do not guarantee better privacy protection.

We then study computation time against privacy property k on the two synthesized vector-action sets ($6 \times engine^B$ and $64 \times drug^B$). As expected, rounding is insensitive to k since it does not have the concept of k. 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 $O(n)$,



(a) Flow Data Cardinality (b) Privacy Property

Fig. 3. Execution Time

the computational complexity here is equivalent to that in Section 4.3. 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:

- For each vector-action set VA_i, mvmdGreedy first partitions it into padding groups based on the prefix of each flow-vector (which has $O(|VA_i|)$ solution).
- 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 algorithms for concrete vector-action sets is also a function of k. These two datasets in our experimental environment satisfy above condition, for example, $26(\lambda) \times 320(k) \ll 2.7m$ for $search^B$. In other words, in such case the execution time 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. Figure 3(b) illustrates the computation time of mvmdGreedy algorithm 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.

5.4 Processing Overhead

Our previous discussions have focused on reducing the communication overhead of padding while ensuring each flow-vector to satisfy the desired privacy property. To implement traffic padding in an existing Web application, if the HTTPS header or data is compressed, we can pad after compression, and pad to the header; if header and data are not compressed, we can pad to the data itself (e.g., spaces of required padding bytes can be appended to textual data). Clearly, the browser's TCP/IP stack is responsible for the header padding, while the original web applications regard the data padding as normal data. An application can choose to incorporate the padding at different stage of processing a request. First, an application can consult the outputs of our algorithms for each request and then pad the flow-vectors on the fly. Second, an application can modify the original data beforehand based on the outputs of our algorithms such that the privacy property is satisfied under the modifications. However, padding may incur a processing cost regardless of which approach to be taken. Therefore, we must aim to 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 4 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 $drug^B$. Note that in this paper our model and algorithm design has focused on minimizing the padding cost only. We consider refining them for reducing both the padding and processing cost as our future work.

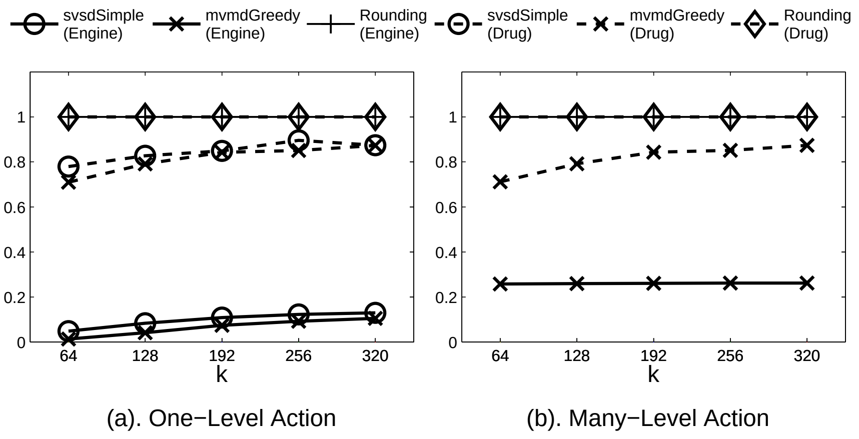


Fig. 4. Processing Cost Overhead Ratio

6 Extension and Discussion

In this section, we first present an extension of our model and then discuss the implementation of our solutions.

6.1 Extension to l-Diversity

We now outline an extension of our model to further show that many existing PPDP concepts may be adapted to address PPTP issues. Specifically, we adapt l-diversity [20] to address cases that no all actions should be treated equally in padding (e.g., some statistical results regarding the likelihood of different inputs may be publicly known).

We first assign an integer *weight* to each action to represent the possibility it occurs. A larger weight indicates that the corresponding action is more likely to be performed. We also slightly change the definition of vector-action set to include the weight information. Then we apply *l-diversity* to quantify the privacy by ensuring the constraint as follows. For each padding group, the summation of weights corresponding to the actions in the group should be at least l times of the maximal weight value in that group.

With the aforementioned revisions, we reformulate the PPTP problem (MVMD problem in Section 3.3) to satisfy *l-diversity* instead. Observe that, the reformulated problem, called *diversity problem*, is simplified to Problem 3 if the weights of actions in a vector-action set are set to be identical and $l = k$. Thus, the *diversity problem* is at least as hard as Problem 3.

Although l-diversity in PPTP shares the same spirit with that in PPDP, algorithms for PPDP cannot be directly applied here, because in PPDP, many tuples may have the same sensitive values, whereas any action in an action-set is always unique, and we assign a weight for each action to distinguish its possibility to be performed from others. The detail of *diversity problem* and its algorithms are omitted due to space limitations.

6.2 Implementation Issues

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 the ceiling padding method lies in the second stage. Thus, we have focused on this stage in this paper. Nonetheless, we have also briefly described how to collect the vector-action sets in Section 5.1 and how to facilitate the third stage in Section 5.4.

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 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 like Google will no longer provide auto-suggestion feature once the query string exceeds a certain length. Third, all the three steps mentioned above are part of the off-line processing, and would only need to be repeated when the Web application undergoes a redesign.

We also note that implementing an existing padding method, such as packet-size rounding, will also need to go through the above three steps if only the 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.

7 Related Work

The privacy preserving issue has received significant attentions in various domains, such as, data publishing and data mining [10, 24], mobile and wireless network [4, 5], social network [11, 22], multiparty computation [21], web applications [6, 9, 26], and so on. In the context of privacy-preserving data publishing, since the introduction of the k-anonymity concept [24, 28], much effort has been made on developing efficient privacy-preserving algorithms [1, 16]. Many other models are also proposed to enhance the k-anonymity, such as l-diversity [20], t-closeness [17]. Recently, differential privacy [12] has been widely accepted as a strong privacy model for answering statistic queries.

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 may extract OpenSSL RSA privacy keys [7]. By differentiating the sounds produced by keys, an attacker may recognize the key pressed [3]. Ristenpart *et al.* discover cross-VM information leakage on Amazon EC2 [23]. Search histories may be reconstructed by session hijacking attack [8]. Saponas *et al.* show the transmission characteristics of encrypted video streaming may allow attackers to recognize the title of movies [25]. HTTPOS are proposed to prevent information leakages of encrypted HTTP traffic in [19]. Timing mitigator is introduced to achieve any given bound on timing channel leakage in [2].

Closest to our work, Chen *et al.* in [9] demonstrate through case studies that side-channel attacks are fundamental to web applications. Our model and solutions provide finer control over the tradeoff between privacy protection and cost. The model section of this paper has previously appeared as a short paper in [18] and now we significantly extend it with problem formulation, algorithm design, and experimental evaluations.

8 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 three algorithms by following the proposed model. Our experiments with both real-world applications have confirmed the performance of our solutions to be superior to existing ones in terms of communication and computation overhead. For future research, we intend to investigate padding approaches for frequently updated vector-action sets, and the possibility of extrapolating the proposed model and approach to mitigate threats of other side-channel leaks.

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, while Kui's research is supported in part by US National Science Foundation through grant CNS-1117811.

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 '10*, pages 297–307, 2010.
3. D. Asonov and R. Agrawal. Keyboard acoustic emanations. *Security and Privacy, IEEE Symposium on*, page 3, 2004.
4. M. Backes, G. Doychev, M. Dürmuth, and B. Köpf. Speaker recognition in encrypted voice streams. In *ESORICS '10*, pages 508–523, 2010.

5. 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.
6. 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.
7. D. Brumley and D. Boneh. Remote timing attacks are practical. In *USENIX*, 2003.
8. C. Castelluccia, E. De Cristofaro, and D. Perito. Private information disclosure from web searches. In *PETS'10*, pages 38–55, 2010.
9. 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.
10. 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.
11. 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.
12. C. Dwork. Differential privacy. In *ICALP (2)*, pages 1–12, 2006.
13. 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.
14. V. Kann. Maximum bounded h-matching is max snp-complete. *Inf. Process. Lett.*, 49:309–318, March 1994.
15. 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.
16. K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient fulldomain k-anonymity. In *SIGMOD*, pages 49–60, 2005.
17. N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE '07*, pages 106–115, 2007.
18. 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.
19. 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*.
20. A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.
21. S. Nagaraja, V. Jalaparti, M. Caesar, and N. Borisov. P3ca: private anomaly detection across isp networks. In *PETS'11*, pages 38–56, 2011.
22. A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy '09*, pages 173–187, 2009.
23. 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.
24. P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. on Knowl. and Data Eng.*, 13(6):1010–1027, 2001.
25. 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.
26. 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.
27. 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.
28. L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.