DIFFERENTIALLY PRIVATE TRAFFIC PADDING FOR WEB APPLICATIONS

TAHER AZAB

A THESIS

IN

THE DEPARTMENT

OF

CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

FEBRUARY 2014

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Taher Azab**

Entitled: **Differentially Private Traffic Padding for Web Applications**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Dr. Amr Youssef _____ Chair

Dr. Chadi Assi _____ Examiner

Dr. Otman Ait Mohamed _____ External Examiner

Dr. Mourad Debbabi _____ Supervisor

Dr. Lingyu Wang _____ Supervisor

Approved Dr. Rachida Dssouli

Chair of Department or Graduate Program Director

_____ 20 _____

Christopher Trueman, Ph.D.,

Dean Faculty of Engineering and Computer Science

Abstract

Differentially Private Traffic Padding for Web Applications

Taher Azab

The wide adoption of Web applications in various sectors of our society, such as government, finance, education, health care, media, etc., has implicitly introduced new security challenges. Among such challenges are side channel attacks that may disclose private user inputs from encrypted traffic. Such attacks might have a serious impact upon user privacy in such applications. In this thesis, we propose a new concept and algorithms that can preserve user privacy in Web applications. In order to achieve this, we define a new privacy model based on a well known concept, namely, differential privacy. The intent is to make padded traffic differentially private such that adversaries cannot infer private user inputs even when they possess prior knowlege about such inputs. At the same time, we intent to achieve a balance bewteen privacy and the incurred communication overhead. In order to demonstrate the usefulness of our model, we implement the proposed algorithms and conduct experiments based on data collected from well known Web applications.

Acknowledgments

"And Allah brought you forth from the wombs of your mothers knowing nothing, and gave you hearing and sight and hearts that haply ye might give thanks." (Holly Coran: Sura/Chapter An-Nahl (The Bees):78)

First and foremost, I would like to thank Allah for providing me with the faith, will and energy to achieve my goals in life. Next, I would like to dedicate this work to my country Egypt, where I got my knowledge basis. I wish to express my genuine thanks and gratitude to my thesis supervisors, advisers and teacher Professors Lingyu Wang, and Mourad Debbabi for their expert guidance and help throughout my research work. Professor Wang's persistent commitment to valuable research, has extremely contributed to finish my thesis. I am proud to have had the opportunity to work under his supervision. I would also to thank NCFTA team members, especially Son Dinh , Amine Boukhtouta and Nour Eddine for their support, encouragements and friendship. I thank all the members of the Concordia Institute for Information Systems Engineering (CIISE) for creating such a dynamic and collaborative environment for research and study. I thank the office staff and the systems supporting staff in CIISE for all their kind assistance. I would like also to thank my great friends William and Mickael for their support and friendship. My warmest thanks and love go to my parents Mrs Mona Ali and Mr Adel Salah who gave me the best support with their sacrifice and love. Finally, My deepest gratitude and love go to my dear sisters Rania and Raghda who with their endless giving love and support throughout my life made everything possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The popularity of Web applications is gaining a momentum in many sectors of our society. Unlike desktop-based applications, Web-based applications require less client-side resources and are easier to develop, deploy and maintain. It is due to the presence of Web browsers on the majority of computers and mobile devices today, which makes it more convenient for both users and service providers to adopt.

Furthermore, such Web applications may also pose new security threats. These threats are result to the fact that Web applications rely on the distrusted Internet as a medium of communication between their client and server sides. As most processing in Web applications is done on the server-side, sensitive data, such as personal emails, online banking transactions and health records, are frequently transmitted through a network of insecure machines.

Various existing security techniques at different levels of the network can provide protection to some extent. However, Web applications may still be vulnerable to novel security attacks, considering the fact that such applications highly depend on the interactions between the client and server sides. Such interactions are relying on the distrusted Internet as the medium of communications. In particular, the recent work by Chen *et al.* [14] demonstrates that highly sensitive data may be disclosed even if Web application traffic is encrypted. Their work also shows that Web applications have serious user privacy

breaches, as a consequence of the so-called side-channel attacks. These attacks are perpetrated through employing unique patterns in the packet sizes and inter-arrival timing to identify Web application states. By retrieving insights into those states, attackers can obtain sensitive information about Web application users. Recently, Shamir *et al.* [25] have unearthed yet a critical vulnerability of the ubiquitous cryptosystem, namely RSA, through a particular strain of side-channel attacks, *acoustic cryptanalysis* [4, 53, 7], which leverages sounds emitted by electronic components. By utilizing this side-channel attack, they have demonstrated that *full 4096-bit RSA decryption keys* can be obtained relatively quickly using normal tools, such as smart phones. This work has raised concerns over the building blocks of the cyberspace and Internet which relies upon RSA to provide secure and private communication channels. Hence, adversaries who can effectively utilize side-channel attacks may pose as serious threats to the security and privacy of any entity connected through the Internet.

Existing security techniques offer little protection against such side channel attacks. To evade eavesdropping and side-channel attacks, Web applications mainly rely on encryption schemes. The most popular approaches to add a layer of protection to Web application communications include *Hypertext Transfer Protocol Secure* (HTTPS) and, for wireless networks, various Wi-Fi encryption schemes. Different encryption schemes have been proposed in the literature in order to mitigate the threat of an eavesdropper in a network, and prevent him from being able to recover original traffic data.

In particular, for a Wi-Fi device to be able to communicate with a wireless access point, an encryption scheme is needed to provide a private channel for such communications. Wired Equivalent Privacy (WEP [1]) is an early Wi-Fi encryption schema. It aims at introducing confidentiality in wireless networks. However, it was revealed to be flawed and obsolete since it allows a potential attacker to infer encryption key and disclose encrypted traffic [10]. Accordingly, Wi-Fi Protected Access protocols (WPA and WPA2 [1]) have been introduced to overcome the inference of encryption key problem. However, such protocols do not hide the size and timing of network packets, and therefore they are still

vulnerable to side channels attacks addressed in this thesis. WPA uses Temporal Key Integrity Protocol (TKIP [1]). The latter has a byte-level granularity since it uses RC4 cipher schema [1]. WPA2 is based on Cipher-block Chaining Mode Protocol (CCMP). The latter uses 128-bit Advanced Encryption Standard (AES) block cipher. CCMP has shown usefulness in securing WPA2, but misses rounding effect [1]. As a result, resulting ciphertext has the same size of the plaintext [14].

In the case of HTTPS, the choice of ciphers is dependent upon each Website to select. Packet sizes could be rounded up to a multiple of a block size in case a block cipher is used, which can potentially prevent a side channel attack based on packet sizes. However, only few Websites choose to use block cipher for encryption, correspondingly, so many popular/important Websites that use the RC4 stream cipher. It is also shown that even when block ciphers are used, the rounding effect is still very little or non existing at all. This is due to the fact that distinction between traffic flows is too large. As result, an eavesdropper can infer sensitive user information only by observing distinct traffic sizes. In the prevailing of these facts, corroborating privacy and confidentiality of Wi-Fi communications between clients and servers in untrusted environment like Internet, is of a paramount importance. In this thesis, we propose a solution through traffic padding technique, that is compatible with Wi-Fi encryption layer. In the sequel, we present a motivating example that puts forward the essence of privacy problems in Web applications.

1.1 Motivating Example

Table 1 illustrates the interaction between users and a well known real world search engine, where traffic packets are observed in terms of their size and direction. Owing to the fact that this search engine provides the so-called auto-suggestion feature, such interaction happens for every keystroke by the user. With each character of a user input typed into the search engine, a b-byte packet is sent from the browser to the server; then two packets are received from the server of sizes 54-byte (Acknowledgment) and s-byte (Suggestions), respectively. Finally the browser sends back a 60-byte (Acknowledgment) packet back to the server.

For the sake of simplicity, we are showing only the packets whose sizes may help to identify user inputs, such as the *s*-byte packet which has a variable size depending on the input, and the *b*-byte packet size. It is important to mention that *b*-byte packets size increments with each input that the user adds. Evidently, an attacker can deduce which input string the user entered based on observation of the traffic pattern sizes (s-bytes and b-bytes), regardless of the encryption. In this work, we assume that an attacker can identify traffic related to Web applications by using techniques such as fingerprinting and de-anonymizing [44]. We also assume that an attacker can map the identified traffic back to user inputs.

User Input	Observed traffic			
a	$b_1 \rightarrow$	$\leftarrow 54$	$\leftarrow 509$	$60 \rightarrow$
cc	$b_2 \rightarrow$	$\leftarrow 54$	$\leftarrow 502$	$60 \rightarrow$
	$b_2+1 \rightarrow$	$\leftarrow 54$	$\leftarrow 473$	$60 \rightarrow$

Table 1: User input example, and corresponding generated traffic

In addition to observations done on the traffic size, the more a potential attacker observes in terms of packets sequences, the easier for him to guess corresponding inputs. To illustrate this, let us consider information collected from the interaction with the search engine. Table 2 summarizes the different pairs input character and size for *s*-bytes packets. Table 3 illustrates size values of the second keystroke, when two successive keystrokes of inputs characters 'a', 'b', 'c' and 'd' are entered. Most first keystroke inputs have distinct *s*-byte packet size. By combining observations done on two consecutive keystrokes, all input strings can uniquely be identified.

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	

Table 2: *s* Value for Each Char Entered as the First

	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 3: *s* Value for Each Char Entered as the Second Keystroke

1.2 Proposed Solutions

A straightforward solution to mitigate the aforementioned side channel attacks is to pad packets so that each input is no longer mapped to a unique size (e.g. random padding, and rounding). However, as this solution provides the needed privacy against such attacks, it might also result in prohibitive additional communication overhead (e.g. 21074% of overhead for a well-known Web application [14]). Moreover, such approach typically aims to maximize, but cannot guarantee the amount of privacy protection. In this thesis, we adopt a different approach, mainly to guarantee a given level of privacy protection, based on a clearly defined privacy model, while trying to minimize the incurred overhead.

There already exist some efforts on privacy preserving traffic padding [34, 33], which aim to improve the aforementioned naive padding approaches by reducing the overhead

incurred by such techniques and at the same time providing privacy guarantee to the traffic padding outcome. The main limitations of those existing approaches is that they are based on a so-called syntactic privacy model which is vulnerable to prior knowledge that may be possessed by adversaries. In this thesis, we adopt the well known differential privacy model [19] in order to remove such a limitation. Considerable attention has been drawn towards finding an approach to provide a privacy guarantee that is not affected by prior adversarial knowledge. Differential privacy has emerged as the most widely accepted solution [18, 22].

Differential privacy defines a new way to query different data sets while keeping minimal (within a certain limit) the chance of identifying any specific individual information from the output. An observer has little evidence as to whether any specific individual is or is not present in the database [27].

In the remainder of this thesis, we propose a solution to the above side channel attack based on the concept of traffic padding (which is to add dummy bytes to packets to increase their sizes such that the sizes corresponding to multiple potential user inputs will become indistinguishable to an adversary) and differential privacy (which is to offer privacy protection even in the face of prior adversarial knowledge about possible user inputs). The main challenge of our solution lies in the fact that existing methods of adding noises to query results for achieving differential privacy cannot be directly applied to our case, because traffic padding can only increase the sizes of packets, but not to decrease them. In other words,as the noise is generated randomly from laplace distribution the generated noise can be negative. Hence, we need to pad the packet with negative noise which is impossible. A negative noise size added to the actual packet size may result in a negative packet size. Therefore, in this thesis, after we have defined the basic differential privacy model for traffic padding, we focus on designing a series of algorithms to deal with the issue of negative noise sizes, and we evaluate their performance using data collected from real world Web applications.

1.3 Contributions

The main contributions of our research effort are summarized as follows:

- This is, to the best of our knowledge, the first work on differentially private traffic padding.
- We tackle the traffic padding problem by using a differential privacy [19] approach, in order to corroborate mitigation of side-channel attacks.
- We formalize traffic padding problem with Privacy Preserving Traffic Padding (PPTP) model [33].
- We devise a concrete algorithm to implement a traffic padding solution.
- We provide solutions to the various limitations and practical issues of this algorithm through a series of extensions to further improve its performance while ensuring the desired level of differential privacy.
- We evaluate proposed algorithms using data collected from real world applications, based on overhead metric, calibrating the amount of padding and trade-off between delaying packets and usability of Web applications.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 reviews necessary background information while Chapter 3 investigates related work. Chapter 4 provides our methodologies including the models and algorithms. Chapter 5 describes our experimental results. Chapter 6 concludes the thesis and gives future work.

Chapter 2

Background

In order to cover the preliminaries of our topic, we provide in this chapter the necessary background information. We first discuss the definition and concepts of *Web applications*, then we review two related concepts, namely, *traffic padding* and *differential privacy*. We explore each concept along with examples to facilitate their understanding.

2.1 Web Applications

The presence of the World Wide Web has changed the way we interact with each other. The Internet is now being extensively used as an instrument to perform many activities of our daily lives. This enormous collection of interlinked documents (i.e., web pages) provides an unbelievably huge amount of information that helps us accomplish our tasks more quickly and easily. On the other hand, software applications have been utilized to accelerate many processes, which would take plenty of time to carry out manually. The rapid development of the Internet and the World Wide Web has brought new opportunities as well as challenges to software developers. As a result, a new term, *Web applications*, arises and completely redefines the purposes of the World Wide Web. The architecture of Web applications has been remarkably evolving in the last few years, from websites consisting of HTML pages and simple interactions to large-scale *n*-tiered applications simultaneously serving millions of users around the world [16].

The meaning of the term *Web application* can vary within different contexts. In one context, a Web application can reside mostly on the client-side (i.e., a web browser) and utilize HTML and JavaScript to provide its functionality while keeping a minimum interaction with the web server. In another context, a complete system, which comprises of multiple web servers, application servers and load-balancing servers along with complex client-side software, is deployed to handle millions of requests concurrently. In the context of this thesis, we adopt the following definition of a Web application: *a stateful software system, which utilizes the* Hypertext Transfer Protocol *(HTTP) as its communication protocol and consists of web servers and client-side software that is executed in web browsers*. The client-side software, which appears to end users as a *web interface*, accepts inputs from users and provides services based on these inputs. Web applications greatly utilize the client-server model for the ease of deployment. While traditional software applications require a great effort for distribution and maintenance, deploying and updating Web applications are almost transparent to the end users. This is mainly due to the ubiquity of web browsers, which are installed on most of consumer electronic devices ranging from PCs and laptops to tablets and mobile phones. The client-server architecture of Web applications also leverages the advantage of the Internet to deliver services to a wide variety of users through HTTP, regardless of their geographic locations.

However, because of the special characteristic of Web applications: they have to be exposed to millions of users. For this reason the security of such application needs to be revised thoroughly to protect the properties of the organizations providing the services as well as the privacy of their end users. Even though a great deal of effort has been put into this matter, there are still grey areas where adversaries can exploit to carry out malicious activities, which may severely violate the privacy of users and damage the organization reputation.

2.2 Side Channel Information Leaks

The recent work by Chen *et al.* shows that highly sensitive data may be disclosed from the encrypted traffic of many popular Web applications [14]. In this section, we briefly summarize their main models and findings in order to motivate our further discussions.

In [14], *side channel information* refers to aforementioned attributes of encrypted traffic, which can reveal information and insights about the communications. Such side channel vulnerabilities of encrypted communications is placing user data confidentiality at risk, especially with the presence of Web applications' unique features such as stateful communications [14].

Consider an online health Web application denoted as $OnlineHealth^A$, where users use it to store their health profiles by choosing an illness condition from a given list by the application [14]. Based on the user choice, the Web application may cause the browser to communicate with the server side of the application in order to update it's state, which enables the user to be able to see related information to the chosen illness. In this example, communications between the client and server side of the application are protected by HTTPS in order to protect user's confidentiality, despite this the communication has several observable attributes that can leak information about user's selection, such as packet sizes and timings [14].

In their recent work [14], Chen *et al.* show how an attacker can leverage the side channel attacks to gain sensitive information and profile people's actual online activities, even when encryption is used (HTTPS and WPA/WPA2). The work also describes the effectiveness of such attacks in extracting sensitive user information when tested against very popular and most relied on Web applications ($OnlineHealth^A$, $OnlineTax^A$, $OnlineInvest^A$, and Google search engine) [14].

The distinct characteristics of Web applications rely on *web flows* as all the program logic and program states. They have to be communicated between the server and client side of the application. This makes Web applications vulnerable to eavesdropping which calls for the need of protection whether it's application layer like HTTPS or network layer

like Wi-Fi encryptions. Through an *ambiguity set reduction* process the attacker tries to learn more about encrypted web traffic that he observes [14].

To make our discussion more precise, we define some notions, where the set of program states are denoted as S, a set that contains the application data on the browser and on the server. The set of acceptable inputs by the application are denoted as Σ, this set describes the user inputs or the backend databases. The transition of state from one to another, could be described as one state receiving some input which triggers it to change into another state. This can be modeled by the following equation:

$$\delta : S \times \Sigma \rightarrow S. \tag{1}$$

When triggered by an input, and the application is at state S, it produces a set of web flow vectors denoted as V, that is the result of interaction between the client and server side of the application. Such web flows have some observable characteristics such as size of packets and their number. To model the observation as function f, we use the following equation :

$$f : S \times \Sigma \rightarrow V. \tag{2}$$

To denote a single $webflowvector$, we use v; this vector is sequence of directional packets that flow between the client and server side, e.g., a browser sends a packet of 50-byte to the server then receives 1024-byte packet are denoted by $(50 \Rightarrow, 1024 \Leftarrow)$ [14].

After modeling the Web application, the objective of the adversary can be formalized as follows. Consider application state to be s_t at any given time t. At state s_t the application accepts an input. The input space is partitioned such that each partition brings the application into unique state that is reachable from s_t, we call these partitions k semantically disjoint sets. For example, different diseases are often grouped by the main symptoms and causes in an online health system. Each K forces the system state into one set of states $S_{t+1} \subset S$.

To know the input set that the application receives in s_t as illustrated in Figure 1, an attacker can do so by looking at n consecutive state transitions initiated by s_t, which triggers the following sequence of vectors (v_t, v_{t+1}, v_{t+2}, ... , v_{t+n-1}). A recursive solution proves

to be possible by starting from s_0, the attacker can infer sensitive inputs for the different states of the Web application. The attacker does not know about the input in s_t, so the size of the ambiguity set is k, when he observe the vector sequence v_t, the attacker knows that this vector can only be produced by transitions to a subset of S_{t+1} denoted by D_{t+1}. This leads the attacker to conclude that such input can only come from k/α sets of the input space. The reduction factor of this state transition is $\alpha \in [1,k)$. The more observations the attacker follows up (v_{t+1}, v_{t+2}, ... , v_{t+n-1}), he can further reduce the new ambiguity set D_{t+1}. This ratio of reduction can be denoted by β, such that $\beta \in [1,\infty)$. Finally the attacker can recognize one of the $k/(\alpha\beta)$ input sets, that contains the actual input [14].

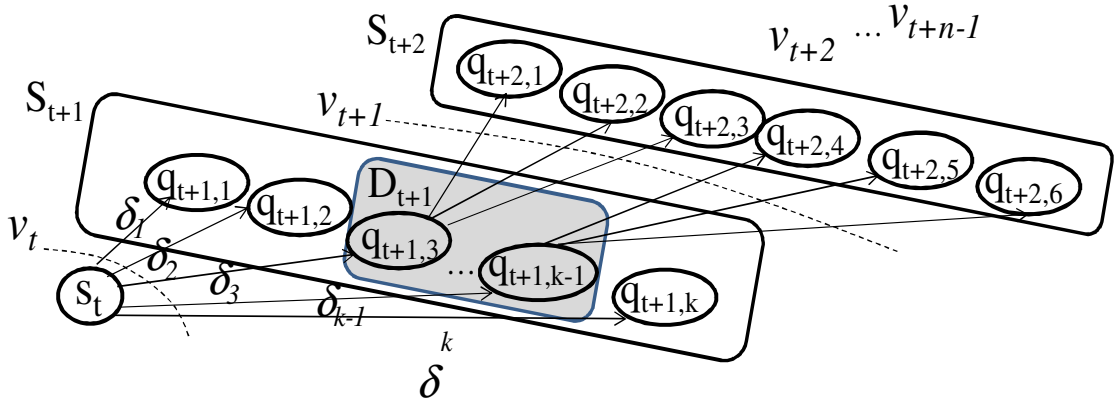

Figure 1: Ambiguity Set Reduction [14]

Moreover an attacker can infer information only relying on the distinct traffic sizes of different web pages. As shown in Figure 2, sizes for JPEG, HTML, and Javascript are collected from five different web pages. For different sizes the means (μ) and standard deviations are calculated (σ) [14].

$OnlineHealth^A$ is an online health system, that enables users to build their health profile through different attributes like their conditions, procedures, and medications. Users can also search doctors based on location and specialty. The system as shown in Figure 3 has several tabs, which already can leak information since clicking on any of them generates a web flow vector (1515 $\pm$ 1 $\Rightarrow$, 266 $\pm$ 1 $\Rightarrow$, 583 $\pm$ 1 $\Leftarrow$, x $\Leftarrow$), where x equals 4855, 30154, 20567, 1773, 2757 and 2299, for Conditions, Medications, Allergies, Procedures,

	JPEG		HTML code		Javascript	
(In bytes)	μ	σ	μ	σ	μ	σ
cnn.com	5385	7856	73192	25862	6453	6684
health.state.pa.us	12235	7374	49917	10591	N/A	N/A
medicineNet.com	3931	2239	49313	14472	22530	28184
nlm.nih.gov	11918	48897	22581	15430	4934	5307
WashingtonPost .com	12037	15122	90353	35476	13413	36220

Figure 2: Size of objects for different web pages [14]

Test Results and Immunizations, respectively. Furthermore the auto-suggestion feature to search for a condition or symptom happens to cause a lot of leaks. This leak is caused due to the distinct web flow vector size that is triggered after each user input. In this case the web flow vector has this form ($253 \pm 1 \Rightarrow$, $581 \Leftarrow$, $x \Leftarrow$), where x represents the size of the suggestion list very precisely and does not change for different users or under different conditions [14].

On the other side, the web flow produced by each keystroke is affected by all the previous keystrokes. So the chance for the attacker to make an inference about the actual keystrokes relies on α and β. There exist 26 x values corresponding to typing a character from a to z as the first input. By examining collected values, it shows that all characters have distinct x value except for the characters h and m. They have the same x value. Inspecting the second character input can even reveal more and provide a better reduction factor α. For example to differentiate between the letters h and m, the attacker needs to consider possible values corresponding to input characters ranging from *ha* to *hz*, and from *ma* to *mz*, which turns out to be all distinct mapping to distinct x values except for *ha* and *ma*. so the more inputs are considered the more reduced set of values we can map the inputs to. This in turns makes it easier for the attacker to guess the user inputs [14].

Figure 4 describes another example of distinct attributes for different system states that causes eventually information leaks. There are 2670 conditions in the OnlineHealth^A

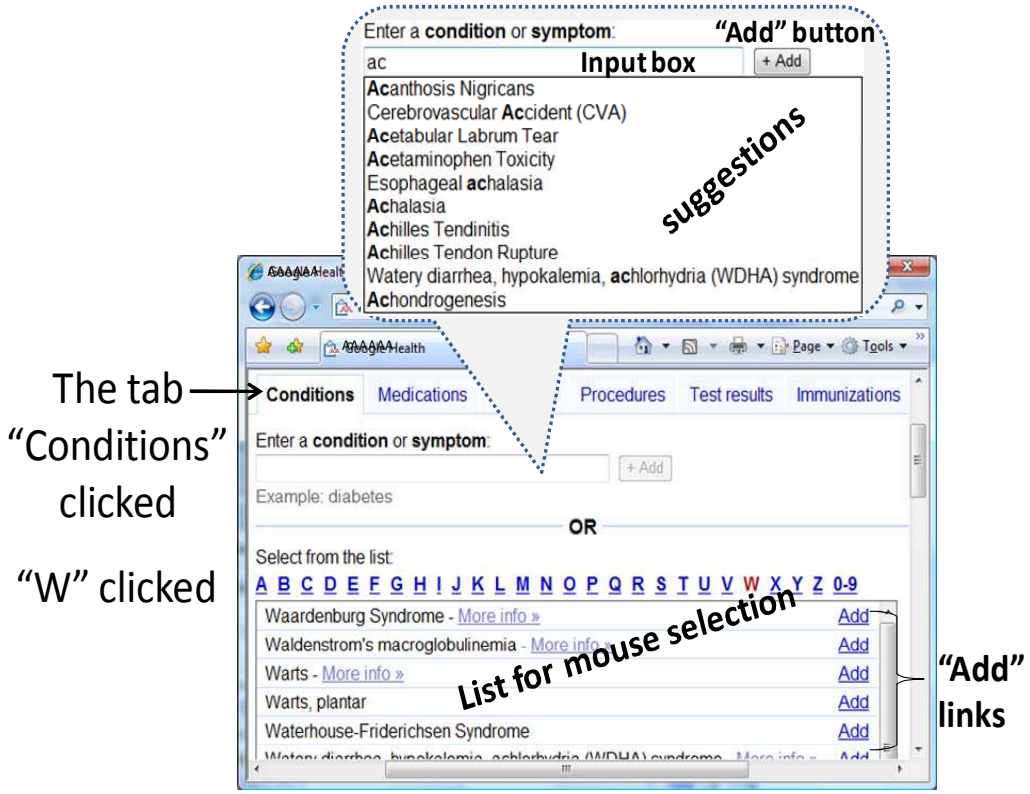

Figure 3: Size of objects for different web pages [14]

system that are distributed among the characters. For example, Figure 3 shows a list of conditions that start with *W* because the user has input *W*. Using this distribution in Figure 4, the reduction power is greatly increased [14].

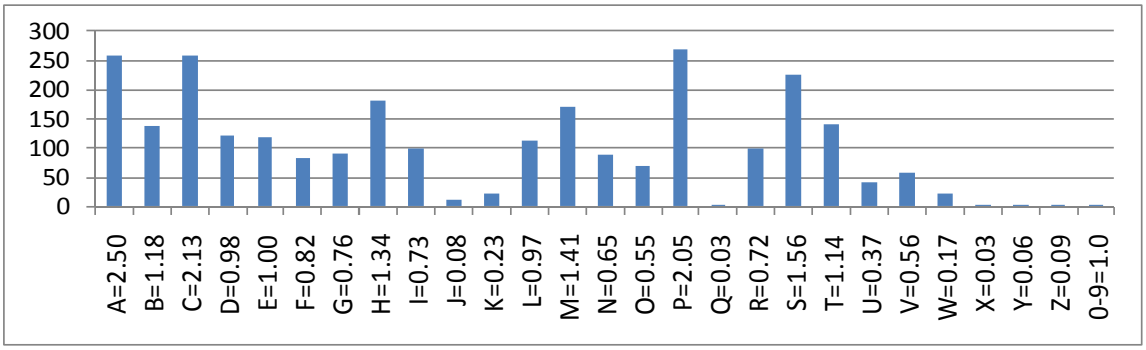

Figure 4: Size of objects for different web pages [14]

Another feature of the OnlineHealth^A system, as shown in Figure 5, is *Findadoctor*.

By entering location and selecting a doctors specialty from the available list, the user is returned a list of results that matches his desired criteria. The location of the user is predictable based on his IP address. When the search is pressed it triggers the following web flow vector (1507 $\Leftarrow$, 270 $\pm$ 10 $\Leftarrow$, 582 $\pm$ 1 $\Rightarrow$, x $\Rightarrow$). In this case x is within the range [596, 1660] [14].

Figure 5: Size of objects for different web pages [14]

2.3 Traffic Padding

In the literature, the word *padding* is commonly used to represent the way of hiding the true nature of traffic volume of web browsing. As it was proved in several existing work [45, 14], monitoring the original data traffic volume can reveal the identity of the user as well as linking him to a particular page he visited, as well as leaking important private information regarding the user activity. An example to such activities that could be exposed by learning traffic sizes, is search activity on interactive search engines. Through examining the size of returned suggestions with every character the user inputs, the attacker can know which query the user has searched. Thus, *traffic padding* is one of the solutions to such side-channel attacks (we elaborate on such attacks in chapter 3). It is used to mask distinguishable data blocks of varying sizes, by creating a uniformly sized (and thus indistinguishable) data blocks [45].

Onion routing is a well known example that uses the traffic padding technique to protect the privacy and confidentiality of data being sent. In its implementation, as presented in [48], the onion proxy receives data from an application to be sent, then the received data is padded to multiples of 128-byte sized blocks, which are then encrypted as they propagate through the connection onion network which lead to indistinguishable blocks

based on the size. Thus, an eavesdropper has no means of distinguishing data based on its size, as he only can see data that is represented in uniform-sized cells (128 bytes).

Moreover padding was used in the literature to mitigate other various side channel attacks, such as the VOIP attack (word spotting) in [51]. In that work the authors explore the tradeoff between padding and accuracy of detecting a specific word based on size. Padding the data to multiples of 128, 256, and 512 blocks, results in overheads of 8.81%, 16.5%, and 30.82%, respectively. The use of padding helped the authors to reduce the accuracy of being able to search for a word corresponding to a specific data size. When using padding to multiples of 128 bits, precision value was 0.16 with a recall value of 0.15, and with 512 bit blocks precision value went down to 0.4 with a recall value of 0.04.

Example 1 *Following the examples discussed in the previous section, in Table 4, we consider an example for padding packets generated by an interactive search engine Web application. The first and the last columns show the s value and corresponding input (the second keystroke). The second column gives one option for padding the packets (although not shown here, there certainly exist many other options). Specically, in that padding option, packets are padded to the maximum value such that their corresponding s values are all identical. This is a natural solution since by padding we can only increase the packet size, but not decrease it. Thus, for an eavesdropper the characters will no longer be distinguishable from each other based on their s values. However, this way of padding have a high cost in case some packets were much smaller (ex: 300-bytes) than the maximum size since it adds too much overhead to each packet.*

As discussed in Example 1, it is notable that there can be a high cost and communication overhead resulting from using this so-called ceiling padding mechanism for padding those packets. Therefore, it is important to find a better way that provides the same effect, which is to make all packets indistinguishable, while at the same time to reduce the overhead to a practical level. Hence, the main objective of this thesis is to find a way of padding packets that can provide the sufficient privacy protection while minimizing the padding cost and overhead.

*s*Value	Padding	(1st Keystroke) 2nd Keystroke
473	509	$(c)c$
477	509	$(c)d$
478	509	$(d)b$
499	509	$(d)d$
501	509	$(c)a$
509	509	$(d)c$

Table 4: Padding Example

2.4 Differential Privacy

Nowadays, it is becoming more and more common to have large databases of highly sensitive information pertaining to different individuals. Potential leakage of such information could violate individuals' privacy, specially with the recent significant increase of usage for these databases in different aspects of our daily life. For example, health systems, financial institutions, search engines, and social networks, etc. all are in possession of huge chunks of personal sensitive information. These organizations face a huge trade off between usability and utility of this sensitive information on one hand (as at some point of time it is needed to make available their data, whether it is for research purposes, legal pressure, or social benefit; for example the medical research can benefit a lot by accessing medical records of health systems/hospitals). On the other hand, they need to guarantee that individual privacy is taken care of and not violated and protect the identities of all individuals affiliated with their databases; for example accessing medical records without any private setting can reveal what exact symptoms or disease a specific individual is suffering from [49, 40].

On the other hand, protecting the privacy of individuals is an obligation of any data owner according to many existing regulations. This implies that users' sensitive data which they have not chosen to reveal need to be kept private. However, due to the research potential that this data can have, there is a need for researchers to be able to access such data

in a certain way. Examples of using such data can be to help researchers to find statistical correlations, such as correlating medical outcomes with risk factors or events; another use is to help an organization to update its strategies through applying appropriate data mining techniques to existing customer data; also the data can be used to publish aggregate statistics [27].

Different attempts to release private data after its anonymization by eliminating obvious identifiers (de-identification) from the data, have proven to fail in protecting individuals privacy as it still leak information and links to specific individuals [27]. Even with no data release, the work by *A. Korolova* [29] shows that internal data mining has caused Facebook to leak users information through leveraging their advertising system.

Considerable attention was drawn towards finding an approach that addresses these privacy issues, and provides a privacy guarantee that is not affected by the attackers previous knowledge. Hence, differential privacy has emerged from the previous work of Dinur *et al.* [18], followed by the work of Dwork *et al.* [22], to provide a complete model for statistical databases privacy. It has been widely accepted as a candidate to substitute the previously proposed partition-based models for privacy preserving data publishing (PPDP).

Differential privacy defines a new way to query different data sets while keeping the chances of identifying the any specific individual information from the output minimal (within a certain limit). An observer has very little evidence about whether any specific individual has, or has not, participated in the database, nor about what values has their data took in this database [27].

Specifically, given any two databases (D, D') that are identical except for any one row/record (the row could also be different, or with an additional row in either of the databases), this pair of databases is referred to as the neighboring databases. Assuming that each row corresponds to one individual, so we have two neighboring databases, meaning that they differ by only one individual participation. A differentially private function K, is a function that selects its output by adding some random factor, such that the probability of any possible output of this function over any two neighboring databases is similar enough [27].

Differentially private mechanisms guarantees that changing a single record in the data set can not affect the outputs to *by much*, by ensuring that the probability of getting any one output should be the same for nearly identical input data sets. On the other hand, it also aims to provide as accurate an output (answer) as possible, under the aforementioned privacy constraint [28]. More formally, we have the following:

Definition 1 Differential Privacy [20] . *A randomized function K provides ε-differential privacy if for all neighboring datasets D and D', and all $S \in Range(K)$*

$$Pr[K(D) \in S] \;\leq\; e^{\varepsilon} \;\times\; Pr[K(D') \in S], \tag{3}$$

where $\varepsilon \geq 0$, Pr is the probability, and the probability space in each case is over the randomness of k.

A differentially private mechanism requires adding randomly generated noise to the output, in order to provide a guarantee of differential privacy. The random noise must be added according to the following property :

$$\forall\; z, z' \;\; s.t. \;\; |\, z - z' \,| = 1 \;:\; Pr[z] \;\leq\; e^{\varepsilon} Pr[z'] \tag{4}$$

where z, z' are any possible outcomes over the datasets D, D', respectively. Moreover, the sensitivity of f over two neighboring databases is defined as the maximum possible difference in the output of f. This is the value that the noise should hide [20, 28] .

Definition 2 *[21] For $f : D \rightarrow \mathbf{R^d}$. The sensitivity of f is*

$$\Delta f \;=\; \max_{D,\,D'} \; ||f(D) - f(D')||_1 \tag{5}$$

$$=\; \max_{D,\,D'} \sum_{i=1}^{d} |f(D)_i - f(D')_i| \tag{6}$$

THEOREM 1 *[21] For $f : D \rightarrow \mathbf{R^d}$, the mechanism K is considered differentially private, if it adds independently generated noise with distribution $Lap(\Delta f/\varepsilon)$ to each of the d output terms*

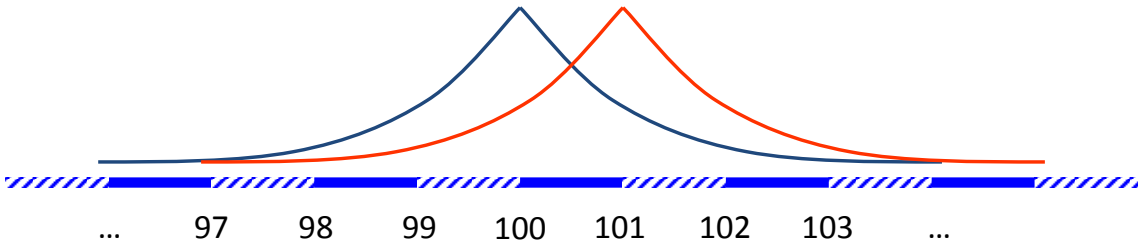

Figure 6: noise based drawn from laplace distribution [20]

Example 2 *Consider a counting query, the sensitivity of this query is 1 ($\Delta f = 1$). Given $\varepsilon = ln\,2$ and the true answer to the query is 100. Referring to Figure 6, the true answer to the query is 100. The distribution on the outputs (in grey) is centered at 100. The distribution on outputs when the true answer is 101 is shown in orange.*

One of the problems of that existing research on differential privacy is trying to address is the magnitude of the added noise, as the less the noise gets the more accurate will be the output of the mechanism, but at the same time privacy of individuals amongst the data set must be guaranteed [18, 21]. Laplacian noise is used by the majority of the differentially private mechanisms, the laplacian noise is applied to the output to ensure that it satisfy the differential privacy conditions.

However, there exist two limitations to the use of laplacian noise in differentially private mechanisms. First, adding Laplacian noise can result in having a negative output which can sometimes be an invalid output, for example for count queries the count output can not be negative. Second, due to the properties of the Laplace distribution, Laplacian noise is unbounded, which could affect the released data making it too large. These two limitations can have a huge negative impact on the data utility [28].

Differential privacy is achieved through adding some randomly generated noise drawn from a laplace distribution (Laplacian noise) with a magnitude proportional to the sensitivity of the query being asked to the database.

Example 3 *Consider two databases that represent the size of packet corresponding to each*

D	
Char	Size
c	15
a	10
t	25

D’	
Char	Size
c	15
a	10
r	20

Figure 7: Character vs Size as Databases

character, such traffic is generated by the server after querying a sequence of input characters. where within this specific application the traffic corresponding to a given character is always with the range {10 - 25}.

As shown in Figure 7, the two tables represent two databases of different words cat and car. It shows as well the packet sizes associated with each. In this specific example, the two words both start with the same two characters which in turn have the same sizes as for both words. However for the last character each word has a different character t and r, and each of the two characters corresponds to a unique size. For an eavesdropper observing the traffic flow sizes and trying to infer what characters corresponds to the sizes he is observing, if he knows the distribution of the different sizes and what character does each of the sizes corresponds to (such information may be obtained by playing with the application with different inputs and observing the size of packets), then it becomes feasible to know what exact word has been the input that triggered such traffic. The ability of the attacker to identify the inputs based on what the sizes of the traffic he observes is considered a side channel information leak.

Looking at this problem from the perspective of differential privacy, we consider that we have a pair of neighboring databases which are D and D', as they are different in only one row. In order to make these two databases differentially private, we should provide a way to query these two tables/databases, such that an attacker seeing the results of query f over D and the same query f over D' could not tell the difference, or link the result to either of the databases.

In order to be able to make the results of the two queries very close to each other, we

need to calculate the sensitivity of a query f over the database D and D', given that f is able to release one character size at a time, one needs to compute the maximum difference/change that can happen to this result. In this example, the maximum change that would be caused by changing one character is the difference between the maximum and the minimum values it can have, wich is 15 in this specific application.

To achieve differential privacy in this case, the server needs to add some noise n to the result r, where this noise should be drawn from a Laplace distribution with magnitude of 15 denoted as lap(15). The result can be represented by this equation : $r = f(D) + lap(15)$. However, since both the noise and the final result may be negative, which does not make sense in the context of traffic padding, such a naive application of the concept as applied to traffic padding will not work. We address this issue in the remainder of this thesis.

Chapter 3

Related Work

3.1 Privacy

Privacy preserving has been addressed by many researchers in various domains and has received significant attention especially in the network and database domains. In this chapter, we review some of the closely related work.

P. Samarati proposed an approach based on the definition of k-anonymity [41], which illustrates how k-anonymity can be provided without violating the integrity and the truthfulness of the information released by using generalization and suppression techniques. P. Samarati also introduced the concept of minimal generalizations to capture the property of the release process to limit distorting the data to the amount needed to achieve k-anonymity. Moreover an algorithm for the computation of such generalization is presented in the paper with the discussion of possible policies to choose from amongst different minimal generalizations.

V. Ciriani et al. focused their work on privacy protection in data mining [15]. Describing the concept of k-anonymity, V. Ciriani et al. illustrated different approaches for its enforcement, then discussed how the privacy requirements characterized by k-anonymity can be violated in data mining and introducing possible approaches to ensure the k-anonymity is satisfied in data mining.

For speaker recognition in encrypted voice streams, M. Backes et al. developed a new

approach for unveiling the identity of speakers who participate in encrypted voice communication only by eavesdropping on the encrypted traffic [6]. They exploited the concept of voice activity detection (VAD) which is widely used for reducing bandwidth by Recently, showing that it creates patterns in the encrypted traffic then they show that these patterns are speaker-characteristic.

K. Bauer et al. focused on privacy of the network protocols that encrypt the entire wireless packets showing that even then these protocols neglect to hide identifying information that is preserved within the wireless physical layer [8]. Furthermore, they exploited this finding by proposing a technique that uses the commodity wireless hardware so that packets are linked to their respective transmitters showing that it degrades user anonymity.

Privacy preserving in social networks has also been studied in many work. P. W. L. Fong et al. worked on delineating the design space of privacy preserving techniques for Facebook-style social network systems, and proposing a formal framework for policy analysis in such systems [24].

In an alternative work [17] G. Danezis et al. proposed a framework to preserve privacy while maximizing the benefit of sharing information in a social network as well as making use of cohesive social group concept from social network analysis. They showed as well that k-anonymity can be used to guarantee privacy in such a real world social network. Another framework [38] is also proposed by A. Narayanan et al. for analyzing privacy and anonymity in social networks, where they have developed a new re-identification algorithm to target anonymized social network graphs, this algorithm mainly based on the network topology.

N. Cao et al. and C. Wang et al. have both focused on outsourced data privacy preserving [12] [50]. The work of N. Cao et al. defined and solved the problem of privacy preserving query over encrypted graph structured data in cloud computing (PPGQ) where they also established a set of strict privacy requirements to create a real secure cloud data utilization system. This work utilized the principle of filtering-and-verification, N. Cao et al. also proposed some techniques to meet the challenge of supporting graph queries using a secure inner product computation achieving with it various privacy requirements. The

work of C. Wang et al. defined and solved the problem of effective yet secure ranked keyword search over encrypted cloud data where in their solution they have used the existing cryptographic primitive order preserving symmetric encryption (OPSE).

S. Nagaraja et al. proposed an algorithm [37] that leverages secure multiparty computation to design a privacy preserving variant of principal component analysis (PCA) that limits information propagation across domains. This algorithm enables ISPs to preserve their private traffic information while allowing them to cooperatively detect anomalies.

Focusing on Web applications, namely Location-Based services, I. Bilogrevic et al. addressed the problem of privacy in one of these services, which is the fair rendez-vous point (FRVP) determination service [9]. They proposed two-privacy preserving algorithms for the FRVP problem, furthermore they analyse and evaluate the privacy of such algorithms in different scenarios as well as evaluating performance by implementing the two approaches on Nokia mobile devices.

In the work [14] by S. Chen et al. they studied the side channel information leak in Web applications traffic and showed that despite the encryption of this traffic information leak is still a realistic and threatens user privacy. Chen et al analysed the traffic produced by some top of the line Web applications showing that an eavesdropper can infer the user queries and his state on the Web application despite HTTPS protection and WPA/WPA2 Wi-Fi encryption. Furthermore, they analysed the challenges of mitigating such threat in Web application developments.

J. Sun et al. addressed the privacy of electronic health record (EHR) systems[43]. In such systems, patients care about their protected health information (PHI) as it contains highly confidential data that needs to be guaranteed proper use and disclosure, especially in case the patient is physically incompetent to retrieve the controlled PHI for emergency treatment. J. Sun et al. proposed a secure EHR system, namely Healthcare System for Patient Privacy (HCPP). The HCPP system is based on cryptographic constructions and existing wireless network infrastructures, allowing it to reliably provide privacy protection to patients, as well as PHI retrieval in emergency situations for life-saving treatment. The

HCPP system prevents PHI access to authorized physicians who can be traced and improperly disclose PHI. The system supports as well efficient and private storage/retrieval of PHI leveraging the support of wireless network access.

In the work of L. Sweeny et al [47] the authors introduced the k-anonymity privacy preserving model as a solution the issue of preserving the privacy for data publishing, and data sharing. Their k-anonymity model provides a scientific guarantee that the individual who are the subjects of the data cannot be re-identified while preserving the usefulness of the data in the same time. L. Sweeny provided a set of accompanying policies to the k-anonymity model for deployment, which helps in mitigating re-identification attacks.

In the context of privacy preserving data publishing, and after the introduction of k-anonymity concept as in the work of L. Sweeny, and the work of P.Samarati, more efforts were directed towards the development of efficient privacy-preserving algorithms. We discuss such algorithms in the following two works of G. Aggarwal et al. [2] and K.LeFevre et al [30]. respectively.

In the work [2] of G. Aggarwal et al., they analysed the problem of releasing k-anonymized tables from a relational database containing personal records. They showed that the k-anonymity problem is NP-hard then they proposed an O(k)-approximation algorithm for the problem that provides improvements to previous best-known O(K log K)-approximation.

In another work K. LeFevre et al. proposed a framework for implementing one model of k-anonymization [30] , called full-domain generalization. The model aims at mitigating the risk of Joining attacks against published microdata and public databases. They also introduced a set of algorithms that aims at producing minimal full-domain generalizations, that performs faster than previous algorithms on real-life real databases. Another contribution of K. LeFevres work was providing a single taxonomy categorization previous models and introducing new alternatives.

As an enhancement to the k-anonymity model, various new models were proposed. Following we will mention two of those latest work. A. Machanavajjhala et al. have proposed the l-diversity model [36]. The main advantages of the l-diversity model is that firstly, it guarantees more privacy and defends against attacker that has background knowledge, and

secondly it solves a known k-anonymity problem where attacker can discover the values of sensitive attributes that have little diversity. A. Machanavajjhala provided as well some experimental analysis and evaluation of the proposed model showing that l-diversity model is practical and can be implemented efficiently.

On the other side, N. Li et al. in their work [31] exposed a number of limitations that exists in the previous l-diversity privacy model. To address these limitations and mitigate them, N. Li proposed the t-closeness, which is a novel privacy notion that requires that the distribution of any sensitive attribute in the overall table should be close to the distribution of that attribute in any equivalence class. For measuring t-closeness requirement the authors used Earth Mover Distance. The authors as well illustrated advantages of t-closeness over previous models, and discussed the rationale for it.

Recently, differential privacy introduced by C. Dwork [19], has been widely accepted as a strong privacy model for answering statistic queries. In her work, C. Dwork showed that the formalization of Dalenius goal along the lines of semantic security is unachievable. The proposed differential privacy is capable of capturing the increased to ones privacy that happens when participating in a database. The differential privacy proposed techniques can be used to achieve any desired level of privacy under this measure. One of the advantages of the differential privacy, is that it can provide extremely accurate information about the database while ensuring high levels of privacy.

3.2 Side Channel Attacks

Researchers have extensively studied various side channel attack leakages in the literature. In this section, we discuss some of those work that are closely related to our research.

Timing attack is a popular side channel attack, where attacker depends on observing the server response time to detect the system state. The work of [11], demonstrated how such side channel attacks exists, by devising a timing attack against OpenSSL. The attacker may extract private keys from an OpenSSL-based web server by measuring the amount of time taken to respond to queries.

D. Asonov et al. were able to differentiate the sounds produced by keys on different devices with a keyboard and recognizing the key pressed [4]. Their attack was employed by using a neural network. In their work they have also proposed some hints on how to design keyboards that would mitigate such attacks. A novel attack that was able to recover 95% of 10 minute input of english text was presented in the work of L. Zhuang et al. [53]. The attack is also based on sound patterns and it used a combination of standard machine learning and speech recognition techniques, including cepstrum features, Hidden Markov Models, linear classification, and feedback-based incremental learning.

Fingerprinting websites is another side channel attack that was studied by X. Gong et al [26]. In the work they demonstrated how an attacker might find out which website a user is accessing remotely without being able to directly observe traffic patterns by using a data mining technique namely, k-nearest neighbor classification. X. Gong relied on exploiting a queuing side channel in routers by sending probes from a far vantage point. The accuracy of such attack at fingerprinting the websites remotely was found out to be 80% after experimenting.

T. Ristenpart et al. [39], using the Amazon EC2 service were able to show that it is possible to map the internal cloud infrastructure. Furthermore they were also able to locate a particular targeted VM, then given the fact that different VMs share the same physical infrastructure T. Ristenpart et al. were able to instantiate new VMs until one shares the same physical infrastructure with the targeted VM, then using this new co-resident VM to launch cross-VM side channel attack to extract information out of the target machine.

Historiographer is a novel attack proposed by C. Castelluccia et al., which is able to reconstruct the web search history of Google users [13]. Using a reconstruction technique, the Historiographer is able to infer search history from observing google personalized suggestions.

E. W. Felten et al, showed that a malicious website can detect some other unrelated web page visited by the user by measuring the time the browser of the victim takes to perform certain operations [23]. Since visiting a web page will cause the browser to build some cache for this page, requesting some parts of this page can reveal whether this page

was cached before and hence visited or not. In the paper, they also described a way of preventing most of these attacks by reengineering web browsers.

T. S. Saponas et al. exposed a vulnerability in Slingbox Pro that causes information leakage for encrypted streaming multimedia [42]. This exposure lead them to be able to determine with high probability the title of the movie that a user is watching on Slingbox by exploiting variable bitrate encoding schemes properties through the transmission characteristics of the encrypted video.

Encryption is often proposed as a tool for protecting the privacy of World Wide Web browsing. However, encryption-particularly as typically implemented in, or in concert with popular Web browsers-does not hide all information about the encrypted plaintext. Specifically, HTTP object count and sizes are often revealed (or at least incompletely concealed). We investigate the identifiability of World Wide Web traffic based on this unconcealed information in a large sample of Web pages, and show that it suffices to identify a significant fraction of them quite reliably. We also suggest some possible countermeasures against the exposure of this kind of information and experimentally evaluate their effectiveness.

On the other hand, efforts have been made on developing techniques to mitigate the threats of such information leakage. As web browsers does not hide all information about the encrypted plaintext. In the work of Q. Sun et al. [45] several countermeasures are suggested against the exposure of identification of encrypted web traffic like Padding, Mimicking and Morphing. The work also provides analysis on the cost of each countermeasure with different parameters, as well as comparison between the various countermeasures.

X. Luo et al. proposed a novel browser-side system called HTTPOS, that prevents inferring information from analysing the encrypted HTTP traffic and offers more scalability and flexibility [35]. HTTPOS uses a suite of traffic transformation techniques that is comprehensive and configurable that allows the browser to defeat information leakage without the need for any server-side manipulation.

A. Askarov et al. investigated techniques that mitigate timing side channel attacks [3]. They introduced as well a set of time mitigators that delay output events enabling the system to achieve any given bound on the timing channel leakage with sacrificing part of

the system performance.

As virtualization in the cloud relies on physical co-residency, recent research have demonstrated that attackers may be able to reach sensitive data through side channels across virtual machines (VMs) that share common hardware. In their work [52] Y. Zhang et al. present an approach to verify the physical isolation of VMs that indicates the exclusive use of a physical machine for a given VM. The work is based on analyzing cache usage, where side channel in the memory cache is used as a defensive and detection tool.

A. Aviram et al. proposed a new approach using provider-enforced deterministic execution to control and eliminate time channels within the same cloud domain [5]. Provider-enforced determinism ensures that any tasks output will not leak any valuable timing information.

Chapter 4

Methodologies

As we have discussed in previous sections, encryption techniques used to protect network traffic are prone to side channel attacks, and can cause sensitive information leakage. This is due to both the poor security design of the used applications and bad choices of the encryption techniques that, inspite of hiding the plaintext in the form of ciphertext, are still preserving many unique attributes of the plaintext in the produced ciphertext. This in turn can lead an attacker into gaining knowledge about what was intended to be secret and protected. We have discussed how such attacks are possible in Chapter 1 and 2 even when Web applications are protected by HTTPS and Wi-Fi encryption schemes like WEP and WPA/WPA2. Hence, the need exists for a different approach that provides the needed protection against such side channel leaks.

In this chapter, we present our approach that aim to address the traffic padding problem in order to corroborate mitigation of side-channel attacks. This is achieved through devising algorithms that adopt the differential privacy approach cite[18]. Previous efforts have been made to improve existing traffic padding techniques [34, 33, 14, 32, 46]. The efforts aim mainly to reduce the overhead of the traffic padding of such techniques while maintaining a robust privacy guarantee to the outcome traffic. In the approach, we propose a novel differentially private traffic padding technique that ensure a stronger level of privacy compared to the existing solutions. Meanwhile, the approach reduce the cost of overhead in the traffic padding. The two objectives achieved in steps:

- Formalizing traffic padding problem with Privacy Preserving Traffic Padding (PPTP) model [31] in section 4.1
- Design two algorithm solutions for the purpose of preserving privacy and reducing the overhead of the traffic padding. The introduced algorithms are discussed in details together with deep analysis of the advantages and limitation. The following are the category solutions :
  - The packets splitting and padding algorithms that use packet splitting to address the negative noise while the padding is dedicated for the positive nose. These algorithms are introduced in Sections 4.2.2, 4.2.3, and 4.2.6.
  - Packet grouping algorithms which mitigate negative of one packet by adding it to the positive noise of the successive packet section 4.2.4 and 4.2.5.

The algorithms are presented in progressive order, to help the reader to follow the limitation we face in each step that lead to the extension, until we reach our final solution.

4.1 The Model

In this section, we first present the traffic padding model including *interaction* between client and server and the *observation* made by eavesdroppers in Section 4.1.1. Moreover, the discussion of the privacy properties of our differentially private traffic padding model is provided in Section 4.1.2. Finally, in Section 4.1.3 we define our distance and padding cost metrics, which we will use to evaluate our proposed solutions by computing the produced padding overhead. For refrence, background on traffic padding and differential privacy is provided in subsection 2.3 , and 2.4 respectively.

4.1.1 Traffic Padding

There exist two main perspectives that have to be considered in order to be able to model the traffic padding problem formally. First, the *interaction* between the user and the server that

describes the user actions (e.g. keypress, mouse click. etc). To that end, the eavesdropper aims at deducing these actions. Second, the *observation* made by the eavesdropper based on the network traffic resulting from the interaction.

For instance, in table 1 the interaction is the user input (a & cc), while the observation to these actions made by eavesdropper are represented by packet sizes. The challenge for the eavesdropper is to infer the user input through observing packet sizes.

Moreover, the table 1 shows how the traffic generated changes based on the order of occurrence of the input. For example, two different observed traffic sequences refer to the same input character c. Such inter-dependent user *actions* (e.g. cc) are modeled as an action-sequence. Eavesdropper maybe combine multiple observations to gain more inference as we discuss in section 1.1. In the sequel, we provide formal definition of the interaction in Definition 3.

Definition 3 (The interaction) *In a given Web application we define :*
- Action a : is a user input that causes traffic (keystroke or a mouse click).
- Action-sequence $\vec{a}$: is a sequence of consecutive actions, which can be keystrokes entered into a search engine or series of mouse clicks through some menu items. $\vec{a}[i]$ will be used to denote the i^{th} action in $\vec{a}$
- Action-set A_i : is the collection of all the i^{th} actions in a set of action-sequences

Example 4 *Back to the table 1, there exist two action-sequences, a and cc, and there exist also two action-sets $A_1 = \{a, c\}$ and $A_2 = \{c\}$*

Definition 4 models three observation concepts namely flow-vector, vector-sequence, and vector-set. Note that a flow-vector is intended to only model those packets that may contribute to identify an action. Also, each action is associated with a $flow-vector$. The latter, consists of all the flows corresponding to a user action. Finally, unlike an $action-set$, a $vector-set$ is defined as a multiset, since it may contain duplicates(flows that may share the same size).

Definition 4 (The observation) *In a given Web application we define :*
- Flow-vector v : is a the sequence of flows represented by integer as the packet size (traffic),

it corresponds to an action a based on the packets it triggers.

- Vector-sequence $\vec{v}$: is a sequence of consecutive flow-vectors, which is resulted by an action-sequence $\vec{a}$, where $|\vec{v}|=|\vec{a}|$. $\vec{v}[i]$ will be used to denote the i^{th} action in $\vec{v}$

- Vector-set V_i : is the collection of all the i^{th} flow-vectors in a set of Vector-sequences, which also corresponds to an Action-set.

Example 5 *Following Example 1, there exist three flow-vectors $v_1 = 509$, $v_2 = 502$ and $v_3 = 473$ that corresponds to the actions a , c (as first keystroke) and c (as second keystroke). There exist also two vector-sequences $\{v_1\}$ and $\{v_2, v_3\}$. Three vector-sets exists as well $V_1 = \{509, 502\}$, and $V_2 = \{473\}$ corresponding to action-sets A_1 and A_2 from Example 1.*

4.1.2 Privacy Properties

For simplicity, we first consider a simplified case where every action-sequence and flow-vector are of length one. In the search engine context considering the simplified case, the action-sequence and the flow vector are results of a user query that is consisting of one character. This case is called the Single-Vector Single-Dimension (SVSD) case. In this simplified case all actions are independent, and each action triggers only one single unique packet that can be used to identify the action.

In this case, we can map a given vector-action sequence $\vec{VA} = \{(v,a) : v \in \vec{v} \wedge a \in \vec{a}\}$ to a table $T(v,a)$ with three attributes, the flow-vector (equivalent to a flow of s-bytes), the action-sequence ($\vec{a}$), and the actual key pressed corresponding to each $\vec{a}[i]$ which is the sensitive value that can leak information about the actual key pressed.

Definition 5 Differentially private traffic . *Given two $\vec{VA}$ sequences that are different just in one (v,a), we define :*

- padding as adding random noise n to each s in the flow-vector $v \in VA$, where this noise is drawn from Laplace distribution

- we say that $\vec{VA}$ satisfies ε-differential privacy if $\frac{Pr[\vec{v} \to \vec{VA}]}{Pr[\vec{v} \to \vec{VA}']} \leq e^{\varepsilon}$.

- the two different $\vec{VA}$, in the differential privacy context will represent D and D'.

Example 6 *In the case of Figure 8, consider the database that logs observed traffic (Vector-Sequence) and maps every flow-vector to the action that triggered this traffic. However, the attacker cannot query the actual key pressed. In this one record case, the attacker can infer information about the specific key that the user pressed just by comparing the results of the query to those in "table 2". As it's known that 491 bytes corresponds to pressing 'x', while 502 corresponds to pressing 'y'.*

To apply differential privacy on such case, we need to find the maximum difference between the flow-vector of two rows. The maximum difference that can happen in this database is to have the character smallest in size being changed to the one largest in size, this maximum difference is called sensitivity (difference between two neighbouring databases). After we have known that sensitive value, noise is drawn out of Laplace function distribution, with variance based on the sensitivity. The real data will be send together (padded) with the noise to guarantee the privacy.

$\vec{VA}$		
Action	Flow-vector	Key Pressed
$\vec{a}[1]$	491 bytes	x

$\vec{VA}$'		
Action	Flow-vector	Key Pressed
$\vec{a}[1]$	502 bytes	y

Figure 8: Database / flow-vector with one record

Discussion It may appear that adopting differential privacy to solve the problem of information leak is not a guarantee of information protection as opposed to encryption. However we are considering only the cases where encryption is already proven to be prone to side-channel attacks. To that respect, we believe that protection can be ameliorated using an extra privacy layer added to the encryption. Practical approach for better confidentiality on Web application can be achieved through hiding the user input among other possible ones. Since the Web applications are easily accessible, an eavesdropper can inevitably learn about sensitive user inputs.

4.1.3 Distance and Cost Metrics

In order to be able to analyse our privacy model defined in Section 4.1.2, two metrics are needed, distance and cost metrics. For the former, we measure the distance between two different $\vec{VA}$ in terms of the packet size difference between the flow vector of each. For the latter, we measure the cost of padding any given $\vec{VA}$ as the distance between the new padded flow-vectors and the original ones, as formalized in Definition 6 and 7, respectively.

Definition 6 Distance. *Given any two equal length vector-action sequences $\vec{VA}_1$ and $\vec{VA}_2$ we define the distance between $\vec{v}_1 \in \vec{VA}_1$ and $\vec{v}_2 \in \vec{VA}_2$ as :*

$$dist(\vec{v}_1, \vec{v}_2) \quad = \quad \sum_{i=1}^{|\vec{v}_{1,2}|} \; (|\vec{v}_1[i] \; - \; \vec{v}_2[i]|)$$

Definition 7 Padding Cost. *Given vector-action sequence $\vec{VA}$ and its padded counterpart $\vec{VA}'$ we define the padding cost of $\vec{VA}$ as:*

$$cost \; = \; dist(\vec{v}\,, \; \vec{v}\,')$$

where $\vec{v} \in \vec{VA}$ and $\vec{v}\,' \in \vec{VA}'$

4.2 Algorithms

In this section, we design a group of algorithms for applying our model to web traffic such that it satisfies our privacy requirement. Inspite of designing an exhaustive list of solutions, our objective is to demonstrate the existence of different possibilities in approaching the privacy preserving padding issue as we show in the following sections.

Discussion. Suppose that it were possible to add negative padding/noise value. The noise can have a negative value since it is derived from Laplace distribution. On the other hand, in the context of traffic padding, padded packets size can only be more than or equal to the original size. Moreover, to be able to pad the packets to a size that comply to our privacy model, the packets need to be padded in a way that keep the size of the padded packet equal to the size of the original packet plus the generated random noise.

As mentioned the possibility of having negative noises, means that the size of padded packets becomes less than the original size. However, we can only increase a packet size by padding, but not to decrease it. This emerge as one of the problems we try to tackle in our algorithms, as it is impossible in reality to send a packet with negative size, such packet cannot exist or be defined. Moreover, our effort deals with the negative noise while trying to maintain the privacy model as close as possible to differential privacy one. This allows us to provide traffic padding solution that guarantees the differential privacy of the released information. In the following, we present a series of algorithms to address this issue. We analyze the advantages and limitations of each algorithm, and then improve it in later algorithms.

4.2.1 Naive Padding Algorithm

In this first algorithm, we demonstrate a naive approach to achieve a differentially private traffic padding. However, the algorithm does not provide a complete solution due to inability of handling the negative noise as we discuss below. Given a packet P, the packet have a unique size $size(P)$ amongst a range of other unique packet sizes, it means an observer can distinguish between different packets just based on their observed size. In order to mitigate this, we use padding to change the size of each packet. By doing so, the eavesdropper fails to infer information about the original packet by looking only to the observed packet size. For this padding to satisfy our privacy model, we pad the original packet with extra n bytes. The n is derived from Laplace distribution.

Example 7 *Given the following sequence of 3 packets and corresponding sequence of random noises respectively are :* $[50, 70, 40]$ *,and* $[-10, 40, -50]$*:*

- *First, packet to be sent will be of size* $(50) + (-10) = 40$*. Then only a packet of size 40 is sent and the rest is neglected.*
- *Second, packet to be sent will be of size* $(70) + (40) = 110$*. The packet will be padded to reach size 110 and then sent.*

- *Third, packet to be sent will be of size* $(40)+(-50)=-10$. *The whole packet is neglected, since we cannot send a packet with negative size.*

Clearly, some limitations exist in this algorithm. The major limitation is the effect of negative noise which leads to loss of data integrity. For instance, in the above example, in the first packet we lose 10 bytes. Moreover, the whole packet can be neglect due to having negative packet size as illustrated in third packet. In the following algorithms we address this limitation.

4.2.2 Padding and Splitting Algorithm A

The padding and splitting algorithm (A) aims at finding a way to solve the problem of information loss of the naive padding algorithm discussed in the previous subsection. The algorithm starts similarly to the naive one; if n is positive, the packet is padded to the size of $size(P)+n$; however, if n is of a negative size, then that size is removed from $size(P)$ and is buffered to be sent later; the value that is sent is of size $size(P)+n$ which satisfies our privacy requirement. In case that $size(P)+n<0$, then nothing is sent and the whole packet is buffered to be sent later. This process is then repeated on the sequence of upcoming packets until everything is sent. Algorithm 1 illustrates the padding and splitting algorithm(A).

Example 8 *Given the following packet and corresponding sequence of random noises :* $[100]$ *,and* $[-20,-10,15]$*:*

- *the packet to be sent will be of size* $(100)+(-20)=80$. *Then only a packet of size 80 is sent and the rest is padded again.*

- *the rest of the packet adding to it noise* $(20)+(-10)=10$.*Then only a packet of size 10 is sent and the rest is padded again.*

- *the rest of the packet adding to it noise* $(10)+(15)=25$.*Then a packet of size 25 is sent and nothing remains to be padded again.*

Algorithm 1: Padding and splitting algorithm A

```
Data: Packet to be send P;
Data: Noise will be added to packet n;
if n is positive then
    while size(P) < size(P)+n do
        pad(P);
    end
    send(P);
else
    if |n| < size(P) then
        new Packet P_b = subpacket(P, size(P) - |n|);
        new Packet P_b = P - P_a;
        send(P_a);
        algorithmA(P_b);
    end
end
```

ACTION

Applying this algorithm to a packet P where $size(P) = 100$ as shown in Figure : 9, the sequence of packets being sent is (P_a, P_{b1}, P_{b2}); an attacker can see the traffic flow which is (80, 10, 25); if the observer notices that this sequence is the response for one user request, then the observer knows that the sum of this traffic sequence will be at the end $(size(P) + n = 115)$.

Now we can see that only the positive noise is what counts (100+15 = 115), because the negative noise only causes part of the packet to be sent later. where the same results can be achieved if only positive noise is chosen and negative noise is ignored.This will not satisfy differential privacy as only the positive part of Laplace distribution is effective.

Assume that the smallest packet size that can be sent is 1. If the last number in the observed sequence is the minimum size as follows, (80, 10, 1), then the attacker will know for sure that the noise added for the last packet is 0, and in this case the real packet size can be revealed by just calculating the sum of the sequence.

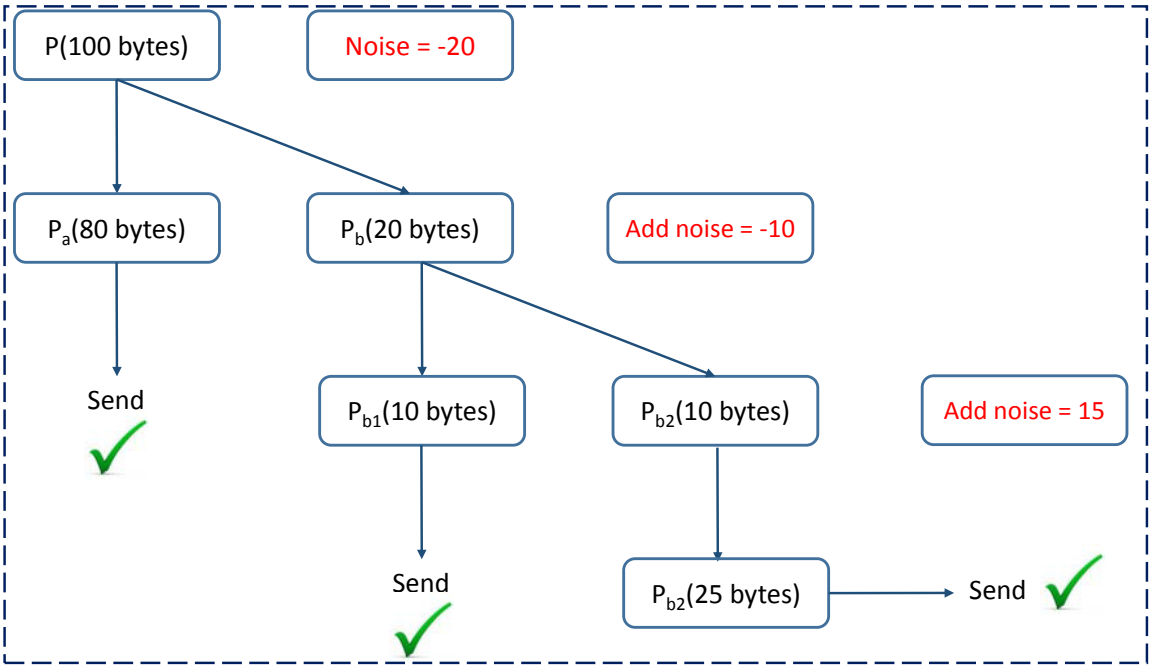

Figure 9: Padding and splitting algorithm A Example

Moreover, the closer the last number in the traffic is to the minimum packet size, the easier it is to predict the original packet size. for example if the last number seen is 3 then it can only be one of the following combinations 1+2, 2+1, or 3+0. Which can allow the observer to eliminate a lot of the ambiguity.

Another limitation of this algorithm is that to we might need to send many padded packets in order to be able to send data contained in the original one.

4.2.3 Padding and Splitting Algorithm B

In this version of the algorithm (B), we try to address one of the limitations of the previous version, which is having to send too many padded packets to deliver the original data of one packet. The algorithm starts with similar instructions. However, after splitting the packet and sending the positive part, the remaining part is added to the next packet instead of being sent by itself as a new packet. Thus it will save some of the overhead caused by treating each packet part as a separate packet. Algorithm 2 describes the padding and splitting algorithm(B).

Algorithm 2: Padding and splitting algorithm B

Data: Packet to be send P;

Data: Noise will be added to packet n;

```
if n is positive then
    while size(P) < size(P)+n do
        pad(P);
    end
    send(P);
else
    if |n| < size(P) then
        new Packet P_a = subpacket(P, size(P) - |n|);
        new Packet P_b = P - P_a;
        send(P_a);
        concatenate(P_b, next(P));
    end
end
```

Example 9 *As showing in Figure 10, given the following sequence of 2 packets and corresponding sequence of random noises respectively are :* $[100, 100]$ *,and* $[-20, 10]$*:*

- *the first packet to be sent will be of size* $(100) + (-20) = 80$*. Then only a packet of size 80 is sent and the rest is added to the next packet.*
- *the second packet to be sent will be of size* $(100) + (20) + (10) = 130$*. The packet will be concatenated with the remaining of the last packet which is 20, then it will be padded with noise of size 10 and then sent.*

Among the limitations of this method, if the noise is negative we have to guarantee that its absolute value is less than the packet size , which is not always possible as the noise is randomly chosen, and a resulting packet size with a negative value is possible (e.g. $n = -100$, and $size(P) = 10$).

Also, adding the remaining part of the previous packet to the new one makes the added value to each packet different than the noise drawn from Laplace, and hence it may not always satisfy the differential privacy requirement.

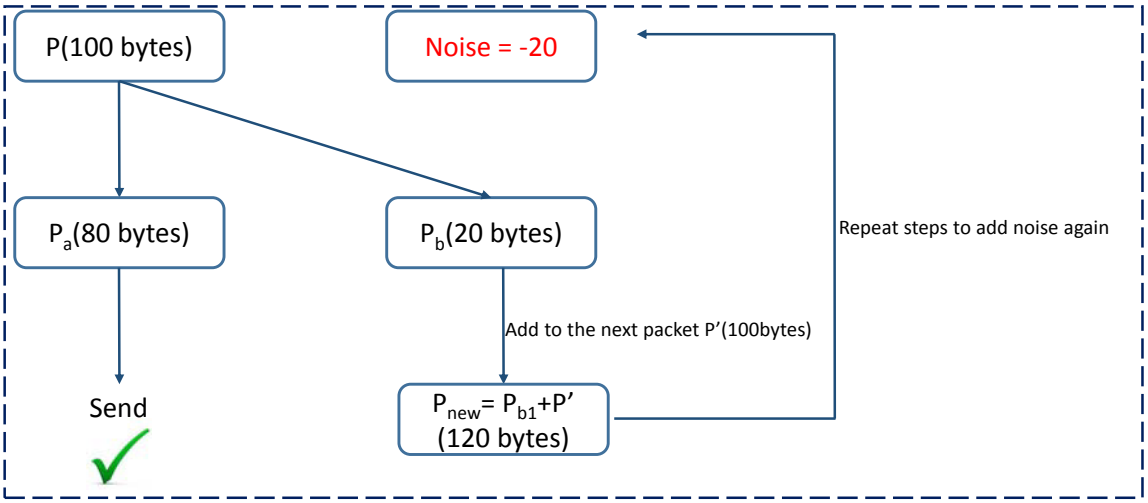

Figure 10: Padding and splitting algorithm B Example

4.2.4 Packet Grouping Algorithm A

In the Packet grouping algorithm (A), we consider a sequence of m packets, for each packet with a noise of size n to be added. Then for each packet if the final packet size ($size(P)+n$) is positive the packet will be sent directly. However if it is negative then the packet will be not be sent and will be grouped with the next packet (which already have noise added to it), until the total amount of noise is positive. In this case we have a chance of removing the negative noise effect by waiting for the next positive noise to cancel it. Thus, we are able to send the complete information without loss. Packet grouping algorithm (A), is described in Algorithm 3.

Definition 8 (Packet Grouping) *Consider a sequence of packets sizes: $size(P_1)$, $size(P_2)$, ..., $size(P_m)$. If a noise is added to each of them according to differential privacy and sensitivity of $size(P_{max}) - size(P_{min})$, regardless of the noise positive or negative. Then, the new sizes $size(P_1)+n_1$, $size(P_2)+n_2$, ..., $size(P_m)+n_m$ will satisfy differential privacy. Hence, the summation of any two of these values should also satisfy the privacy of the packets.*

Example 10 *Given the following sequence of 3 packets (Figure 11) and corresponding sequence of random noises respectively are :* $[100, 80, 120]$ *,and* $[-10, 15, 10]$*:*

Algorithm 3: Packet grouping algorithm A

Data: List packets to be send $P_1....P_m$;

Data: Queue to save packets not send yet *Queue*;

```
1 totalNoise = 0;
2 foreach packet P do
3     n = generate_noise();
4     totalNoise = totalNoise + n;
5     Queue.add(P);
6     if totalNoise >= 0 then
7         sendPackets(Queue, all, noise = totalNoise); // till the queue is empty
          totalNoise = 0;
8     end
9 end
```

- *the first packet to be sent will be concatenated with the next packet since the noise (-10) is negative.*
- *the second packet to be sent will be the concatenation of* 100 *and* 80 . *since the total of the noise for the two packets is positive* $(-10)+(15)=5$*, a big packet will be sent of size* $100+80+5=185$.
- *the third packet to be sent will be of size* $(120)+(10)=130$. *Then only a packet of size 130 is sent since noise is positive.*

As negative noise is not really possible in padding, we wait until such negative noise is cancelled by positive noise of the next packets since the mean of Laplace distribution is around 0, and the probability of getting random positive equals that of getting a negative one equals 0.5. Then we group the packets and send them together as a single big packet.

Although each group of packets is essentially a vector-action sequence $\vec{VA}$, we are not adding noises directly to this $\vec{VA}$. Noises are added together for each group of packets. This allows positive noise to reduce the effect its negative counterpart. In some cases an

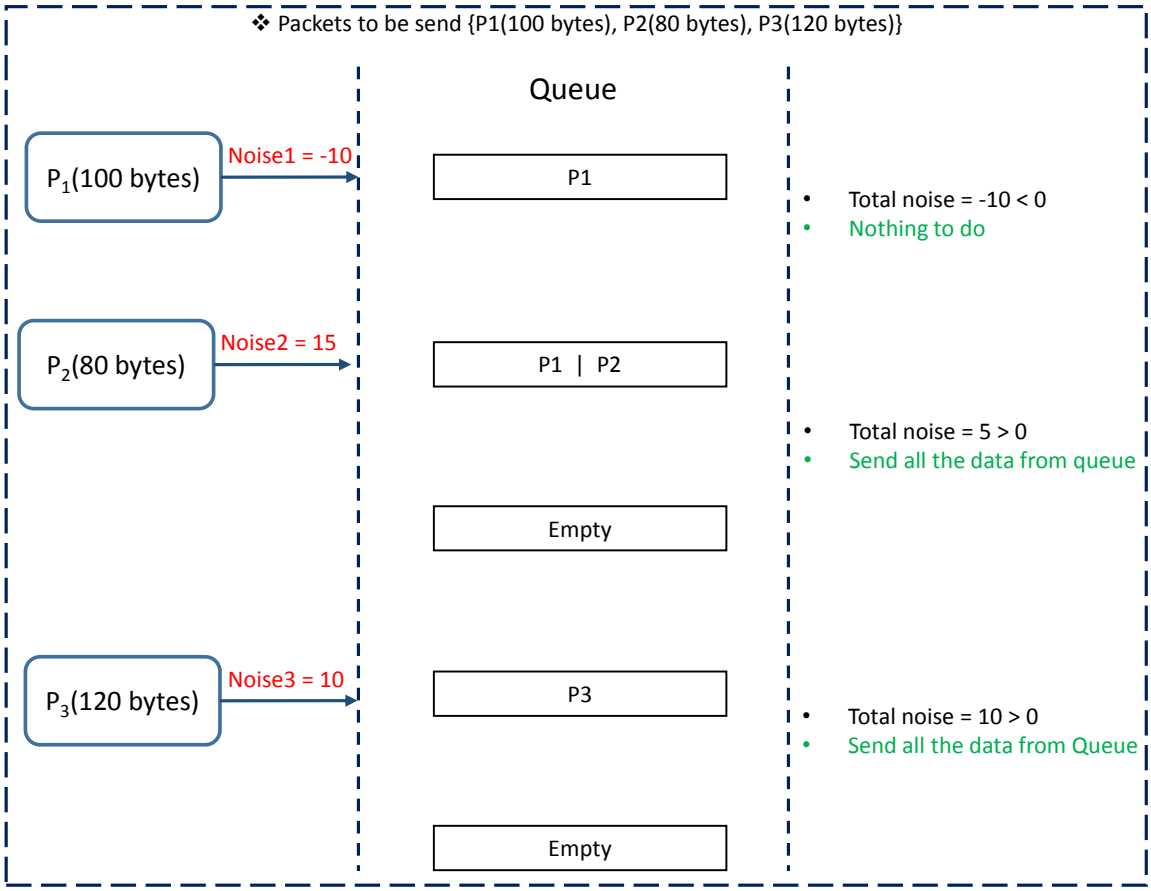

Figure 11: Packet grouping algorithm A Example

observer can still infer each padded packet size from the grouped sizes, but the padded sizes in general satisfy our differential privacy requirement. That is, if it is safe to tell attackers about any of these sizes $size(P_1)+n_1, size(P_2)+n_2, ..., size(P_m)+n_m$ then it is certainly safe to tell them about any summation of these values.

Many limitations do exist with such an algorithm, Firstly, the utility loss due to grouping packets. Secondly, the attacker can always infer about the noises based on the above reasoning; that is if he knows that two packets are grouped together he would know that the first packet noise must be negative, and $n_1 + n_2 \geq 0$.

Finally, the *boundary problem* limitation, which can happen if the observed grouped packet size is twice the minimum packet size. then the attacker will know that both packets have same size which is the minimum size and the total added noise for the two grouped

together is equal to 0 (e.g. attacker see a grouped packet of size 2, there is only one combination possible which is 1+1).

4.2.5 Packet Grouping Algorithm B

Packet delay is one of the problems in the previous algorithms. This delay effect happens due to the surplus of negative noise compared to its positive counterpart. Packets are added to the delay queue as long as the total summation of noise is negative. To tackle this problem we propose in this section algorithm (B) that is an extension to the previous algorithm. The new algorithm deals with the increase of the delay queue size by sending a packet of size P_{max} as long as the summation of packet sizes in the queue is more than P_{max}. Afterwards, the part that is sent is removed from the queue. At the end of the algorithm run, if there still remains pending data in the queue, it will be fitted into packets of P_{max} and sent to the network. The rest of the algorithm remains as the previous algorithm (A). This modification ensures that the delay queue will never get too huge, and consequently the packets will not be delayed for as long as before, even if the noise was always negative (Algorithm 4).

Example 11 *Given the following sequence of 3 packets (Figure 12) and corresponding sequence of random noises respectively are :* $[100, 80, 120]$ *,and* $[-20, 15, 10]$ *and a* $P_{max} = 120$*:*

- *the first packet added to the noise will be of size* $(100) + (-20) = 80$*. Then nothing is sent and 100 is added to the queue since the queue size is less than* P_{max} *and noise is negative.*

- *the second packet to be sent will be the concatenation of* 100 *and* 80 *. Since the total noise* $= -5$ *is still negative nothing is sent and 80 is added to the queue. However, the queue size* $= 180$ *is now bigger than* P_{max}*, therefor, 120 bytes of the queue data will be sent.*

- *the third packet to be sent will be the concatenation of* 60 *and* 120 *. The fact that*

Algorithm 4: Packet grouping algorithm B

```
Data: List packets to be send P_1....P_m;
Data: Queue to save packets not send yet Queue;
totalNoise = 0;
P_max : maxSize(P_1....P_m);
foreach packet P do
    n = generate_noise();
    totalNoise = totalNoise + n;
    Queue.add(P);
    if totalNoise >= 0 then
        sendPackets(Queue, all, noise = totalNoise); // till the queue is empty
        totalNoise = 0;
    end
    while size(Queue) >= P_max do
        sendPacket(Queue, P_max);
        subtract(Queue, P_max);
    end
end
```

total noise $-20+15+10=5$ *is positive, all the data in the queue will be send and the queue is emptied.*

4.2.6 Packet Splitting and Merging Algorithm

This section shows how to deal with negative noise. As discussed previously, we pad the packet with noise of a size that is randomly drawn out of Laplace distribution. This implies that the probability of getting positive noise and that of getting negative noise are equal.

The question is of course how a negative noise can be realized. Hence, packet splitting emerges as a strong candidate for solving the problem. In this section, we introduce a new

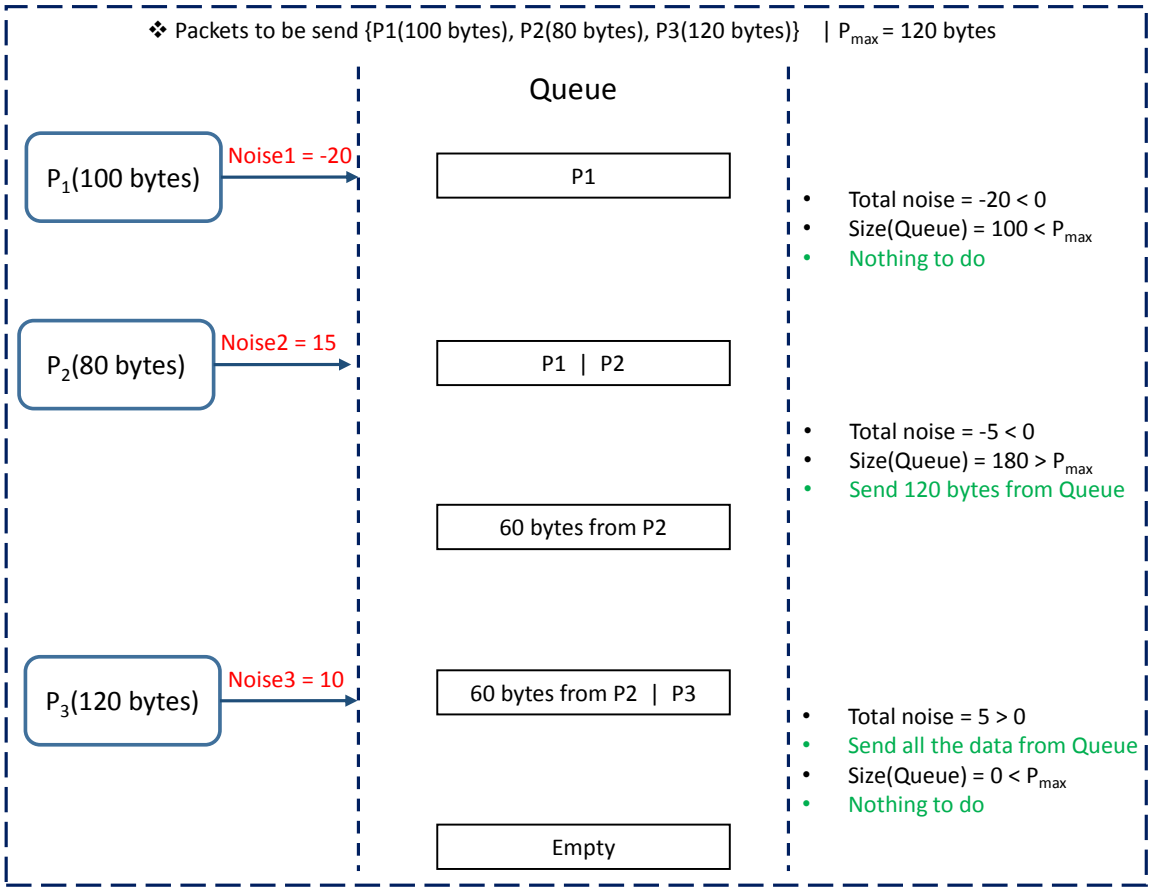

Figure 12: Packet grouping algorithm B Example

algorithm that utilizes packet splitting to achieve the needed privacy requirements. This algorithm could be considered an evolution of the algorithms introduced in Section 4.2.2 and Section 4.2.3. The algorithm works by computing each time the size S that we have to send which is always determined by as $S = size(P) + n$ where P is the packet, n is the noise randomly chosen from Laplace distribution. The value of S is then treated as the size of the container that we will use to send information. Then the size S is compared to the size of packet that needs to be sent. In case that the size of the packet is smaller than the size that we have to send; the packet is then padded to reach that size. However, if the size of the packet to be sent is bigger than S the packet is splitted into two parts one part is equal to the size S, the other part is equal to the rest of the packet denoted by C (i.e. the carryover amount of the packet) which is held and added to the next packet to be sent.

As explained in Section 4.2.3, Algorithm 2 advantage over Algorithm 1 is adding the packet carry resulting by negative traffic to the next coming packet. Algorithm 5 is similar to Algorithm 2 in this part, however, the difference lies in how this carry is handled. For instance in Algorithm 2 the carry is concatenated to the next packet then noise is added such that the size to be sent is equal to the size of concatenated size plus the noise.However, in this algorithm, regardless of the carry and size of concatenated packet, the size to be sent

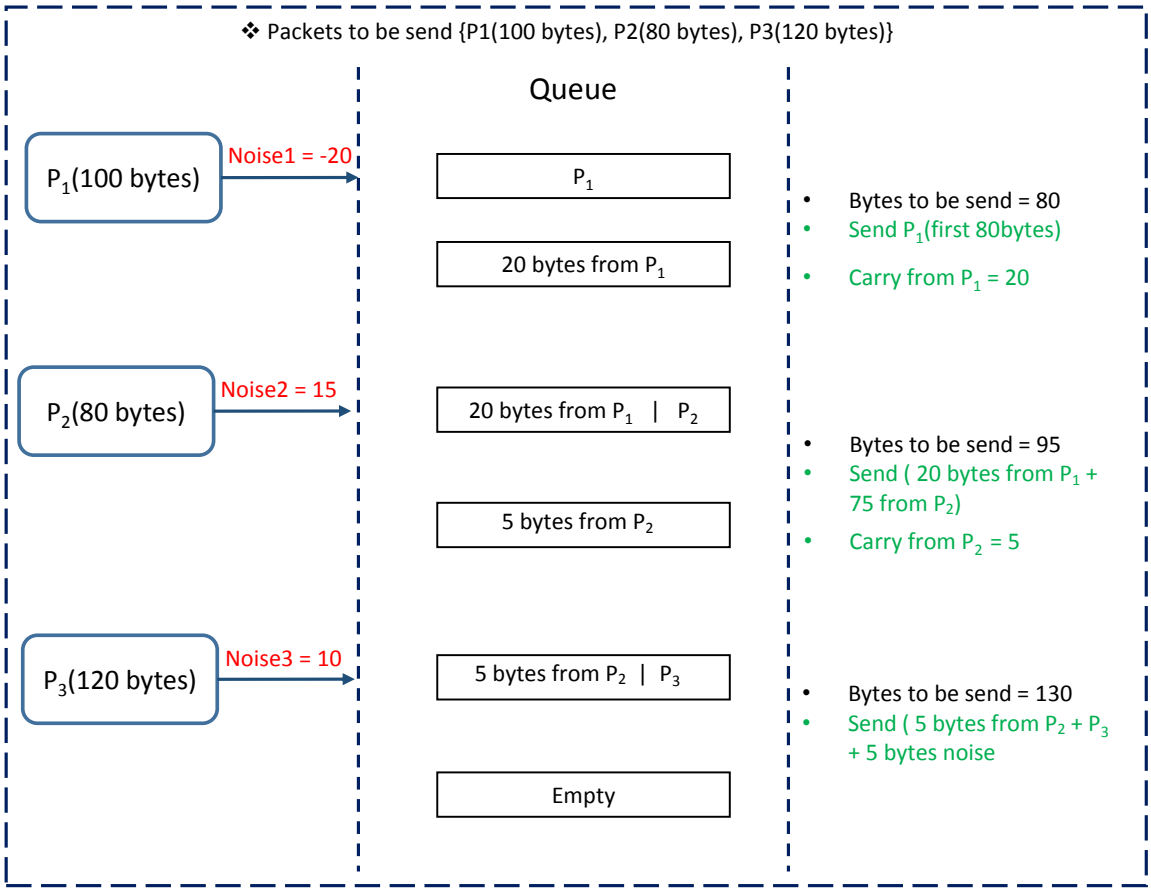

Figure 13: Packet Splitting and Merging Example

Example 12 *Given the following sequence of 3 packets (Figure 12) and corresponding sequence of random noises respectively are :* $[100, 80, 120]$ *,and* $[-20, 15, 10]$ *and a* $P_{max} = 120$*:*

- *the first packet added to the noise will be of size* $(100) + (-20) = 80$. *Then* 80 *bytes of data will be send and* 20 *bytes will be save on the queue.*

- *the second packet added to the noise will be of size* $(80) + (15) = 95$. *Since the carry from previous packet is* 20 *bytes, then the data that will be send is the carry concatenated with* 75 *bytes from the second packets. As result,* 5 *bytes from second packet will be save on the queue.*

- *the third packet added to the noise will be of size* $(120) + (10) = 130$. *Therefor, 130 bytes will be send including* 5 *bytes of the carry,* 120 *bytes the new packet, and* 5 *bytes as noise.*

Algorithm 5: Packet Splitting and Merging Algorithm - Part 1

```
Data: List packets to be send P_1....P_m;
// C = the carry from previous packet.;
// S = size to be send;
C = null;
foreach packet P do
    P_x = concatenate(C, P);
    //1- determine the size to be send
    S = size(P) + n;
    //2- prepare the data to be send
    if size(P_x) < S then
        while size(P_x) < S do
            pad(P_x);
        end
    else
        if size(P_x) == S then
            send(P_x);
        else
            // size(P_x) >= S;
            new Packet P_a = subpacket(P_x, S);
            C = P_x - P_a;      //will be process with the next packet
            send(P_a);
        end
    end
end
```

Algorithm 6: Packet Splitting and Merging Algorithm - Part 2

```
1 while size(C) >= P_max do
2 |   new Packet P_a = subpacket(C, P_max);
3 |   C = C - P_a;
4 |   send(P_a);
5 end
6 if size(C) > 0 then
7 |   pad(C, P_max);
8 |   Send(C);
9 end
```

Chapter 5

Analysis and Evaluation

In this section, we first analyze some practical issues of our approach that was proposed in the previous section. Further we present experimental results.

5.1 Analysis of Practical Issues

One limitation when padding using randomly selected noise, is that this noise can be negative. The designed algorithms shows possible ways to mitigate the effect of this negative noise. However, one practical issue that remains, is when this negative noise happens at the last packet to be sent. Due to this issue only one part of the packet can be sent, and remains the other part of the packet that is delayed due to the negative noise. We refer to this problem as the *last packet problem*.

Example 13 *Given the following sequence of noises* $[5,-3,-1]$ *, the result of applying these noises respectively, will cause the delayed carry in this case to be of size* 4*. That is the result of the negative noise accumulation with no positive noise following to cancel it (*$((-3)+(-1)=(-4))$*.*

5.1.1 Tradeoff Utility against Delay

Definition 9 (Utility) *The utility is defined as the server ability to send packets without delays, regardless the size of the carry that has been caused by previous negative noise which will have to be added to the next packet to be sent.*

Definition 10 (Delay) *The Delay is the mechanism where packets are not sent at once after padding, instead they are sent in groups where the carry can be shared amongst packets of the same group with positive noise, this mechanism is used to mitigate some of the negative noise that can be added to the next packets, the value of the delay is the number of packet within one group that have to be sent together.*

Example 14 *Given a Web application that needs to send 5 packets [P_1,P_2,P_3,P_4,P_5] , the server will behave as follows :*

- *with Delay* $= 1$*, each packet will be sent without any wait for next packets*
- *with Delay* $= 2$*, packets will be sent in groups of 2, [P_1,P_2] will be sent together, then [P_3,P_4], then [P_5].*
- *with Delay* $= 3$*, packets will be sent in groups of 3, [P_1,P_2,P_3], then [P_4,P_5].*
- *with Delay* $= 4$*, packets will be sent in groups of 4, [P_1,P_2,P_3,P_4], then [P_5].*
- *with Delay* $>= 5$*, all packets will be sent together, [P_1,P_2,P_3,P_4,P_5].*

Delay can be used to regulate the negative noises with the positive noises before sending packets. Revisiting Example 13, if delay value of 2 is used, the first two noise values 5 *and* -3 can cancel each other resulting in total noise value of 2. The two corresponding packets will be then padded with total size equal to this total noise value. The last packet is then sent separately, as there are no more packets to be sent in this case.The resulting *carry* will be of size (-1) instead of (-4). Using delay value of 2 lead to minimize the *carry* value. Moreover, choosing a higher delay value (3 or more), causes total mitigation of the

negative noise, as the total summation of the three noise values is positive. This leads to a *carry* value equals to zero. Another effect of delay is the reduction of the total overhead, the positive noise is used to mitigate the negative noise instead of adding more size to be padded(e.g. First noise (5), was reduced to 2 after grouping it with the second noise (-3)).

For different Web applications, the choice can be made either to have less carry value and total overhead by setting a delay value, or to have more utility and send packets with less delay. Thus, it depends on the requirements of each specific application, and the different scenarios that needs to be handled. Consequently, there will be always a tradeoff between the utility and the delay.

Now that we have discussed such a delay versus utility tradeoff through the examples, we now revisit the previous Algorithm 5. Algorithm 7 describes how we extended Algorithm 5 to add the delay factor with the needed modifications.

Therefore, the padding works in a similar way as in Algorithm 5, where the differences lie in the way packets are sent, as we have introduced the delay factor here. Thus, on the cost of utility, Algorithm 7 provides less overhead using the delay factor to compensate partially the effect of the negative noise with the possible occurrence of positive noise of the neighbouring packets.

5.1.2 Noise Limit Calibration

One of the improvements that can be made to the way noise is added in our algorithms, is to find a way to control the noise. Since the laplace noise is not bounded, it can have any value between $-\infty \leq noise \leq \infty$. and when added to the packet size, it can give either negative or a huge packet size both of which are not desirable. In order to control this, we set a minimum and a maximum value P_{min} and P_{max} respectively, such that $P_{min} \leq (P + noise) \leq P_{max}$ which implies that the packet size to be sent after adding the noise cannot be less than P_{min} and cannot be greater than P_{max}.

Example 15 *These are some examples of how noise calibration works given a packet of size* $P = 30$ *where* $P_{min} = 10$ *and* $P_{max} = 50$

- $n = 1000$, $((P+n) = 1030) > (P_{max} = 50)$, $(P+n)$ *trimmed to be 50 , final n value is limited to 20.*

- $n = -25$, $((P+n) = 5) < (P_{min} = 10)$, $(P+n)$ *incremented to be 10 , final n value is limited to -20.*

- $n = 19$, $((P+n) = 49) < (P_{max} = 50)$, $(P+n)$ *does not need to be changed since it lies within the limit.*

Finally, the values of P_{min} and P_{max} do not have to be the same for all cases, it can be adjusted based on each application/network limits and needs. For example, if an application/network is not capable of sending more than 50 per packet, then the limit (P_{max}) can be set to this value to ensure that padded packet sizes will not exceed 50. Same adjustment can be applied on P_{min} if there is a specific value for the packet size that the application cannot send less than. Furthermore, even if the application is capable of sending any packet size, the limits could be set just to avoid too much overhead and improve the efficiency of the padding process since the more difference there is between the maximum and the minimum size more overhead will occur.

Discussion.Given the following standard differential privacy algorithm where $f(D)$ is a query over database D, n is a randomly drawn noise out of Laplace distribution with variance equal to (Δf), and r is the result of adding the noise to the real output, then the second equation should behold true.

$$f(D) + n \rightarrow r \tag{7}$$

$$\frac{Pr[r \rightarrow D]}{Pr[r \rightarrow D']} \leq e^{\varepsilon} \tag{8}$$

In the suggested splitting algorithm, there are three cases for the result r as follows,

$$f(n) + n = r \begin{cases} r = P_{min} & \text{if } r < P_{min} \\ r = r & \text{if } P_{min} \leq r \leq P_{max} \\ r = P_{max} & \text{if } r > P_{max} \end{cases}$$

if $P_{min} \leq r \leq P_{max}$ then it already satisfies the the differential privacy equation as the result has not been it changed. in the other two cases, if $r < P_{min}$ or if $r > P_{max}$ since $\frac{Pr[r<P_{min} \to D]}{Pr[r<P_{min} \to D']} \leq e^{\varepsilon}$, and since $r < P_{min}$ is replaced by P_{min}, then

$$\frac{Pr[P_{min} \to D]}{Pr[P_{min} \to D']} \leq e^{\varepsilon} \tag{9}$$

same applies for $r > P_{max}$.

Assume that with every $r < P_{min}$, we will send a packet of size P_{min} but with letting everyone know the original padded size (r). then our algorithm will satisfy the differential privacy. However if we hide r from the attacker and always show him only P_{min} (similar to naive solution) then his knowledge will decrease relatively. which means that we even achieve more privacy than normal differential privacy by doing this, same applies for $r > P_{max}$ as well. Hence the three possible cases satisfy the differential privacy equation.

Algorithm 7: Packet Splitting and Merging with Delay Queue

Data: List packets to be send $P_1....P_m$;

Data: Delay Queue to buffer packets *DelayQueue*;

Data: Delay value to buffer packets *DelayValue*;

```
1  C = null;
2  while more packets needs to be sent do
3      totalNoise = 0;
4      for DelayValue do
5          addToDelayQueue(P);
6          totalNoise = totalNoise + generateNoise();
7      end
8      sizeToSend = size(DelayQueue)+totalNoise; addToDelayQueue(C);
9      sendPackets(DelayQueue, sizeToSend)
10     if size(DelayQueue) ¿ sizeToSend then
11         C = C + DelayQueue - sizeToSend
12     else
13         C = 0;
14     end
15 end
16 while size(C) >= P_max do
17     new Packet P_a = subpacket(C, P_max);
18     C = C - P_a;
19     send(P_a);
20 end
21 if size(C) > 0 then
22     pad(C, P_max);
23     Send(C);
24 end
```

Algorithm 8: Packet Splitting and Merging With Noise Calibration - Part 1

Data: List packets to be send $P_1....P_m$;

Data: Queue to save packets not send yet *Queue*;

```
// C = the carry from previous packet.;
// S_min is minimum allowed packet size.;
// S_max is maximum allowed packet size.;
// S = size to be send;
C = null;
foreach packet P do
    P_x = concatenate(C, P);
    //1- determine the size to be send
    S = size(P) + n;
    if S < S_min then
        S = S_min;
    else
        if S >= S_max then
            S = S_max;
        end
    end
    //2- prepare the data to be send
    if size(P_x) < S then
        while size(P_x) < S do
            pad(P_x);
        end
    else
        if size(P_x) == S then
            send(P_x);
        else
            // size(P_x) >= S;
            new Packet P_a = subpacket(P_x, S);
            C = P_x - P_a;      //will be process with the next packet
            send(P_a);
        end
    end
end
```

Algorithm 9: Packet Splitting and Merging With Noise Calibration- Part 2

```
while size(C) >= S_max do
    new Packet P_a = subpacket(C, S_max);
    C = C - P_a;
    send(P_a);
end
if size(C) > 0 then
    pad(C, S_max);
    Send(C);
end
```

5.1.3 Extending the example

In this scenario one database D will be considered, to be able to study the way noise is added with our algorithm. Range of possible packet sizes is described as R where R is any value between $S_{min} \rightarrow 50$ and $S_{max} \rightarrow 70$. P_{last} is padded to be of size S_{max}.

D		
Action	Flow-Vector	Key Pressed
P_1	68	p
P_2	51	a
P_3	70	c
P_4	55	k
P_5	53	e
P_6	60	t

Table 5: The packets and their sizes

In the delay of 2 packets case, packets are sent in groups of two. Whenever a packet needs to be sent, noise is added to this packet . but the packet is not sent till we have a second packet. Same method is applied to add noise to the second packet and then if possible some of the positive noise of one packet cancels the negative noise of the other. Packets are sent as follows $([P_1, P_2], [P_3, P_4], [P_5, P_6])$.

Packet	P_1	P_2	P_3	P_4	P_5	P_6	P_{last}
Carry	0	18	0	0	5	0	10
PacketSize	68	51	70	55	53	60	0
Noise	-20	40	30	-10	30	-15	60
EffictiveNoise	-18	19	0	-5	17	-10	60
OverHead	0	1	0	0	12	0	60
Sent	50	70	70	50	70	50	70

Table 6: With no delay

Packet	P_1	P_2	P_3	P_4	P_5	P_6	P_{last}
Carry	0	18	0	0	5	0	0
PacketSize	68	51	70	55	53	60	0
Noise	-20	40	30	-10	30	-15	0
EffictiveNoise	-18	19	0	-5	17	-10	0
OverHead	1		0		2		0
Sent	50	70	70	50	70	50	0

Table 7: With Delay of 2 packets.

As we can see in the case of $[P_5, P_6]$, due to the delay the negative noise of P_6 is mitigated by positive noise of P_5, and the carry will be 0 instead of 10.

5.1.4 Sensitivity Calibration

To further improve our mechanism (Algorithm 5), and to reduce the overhead, we can manipulate the sensitive value of the query. Since the size of traffic resulting by executing an action a depends on the order of this action in the action-sequence $\vec{a}$, which in turn means that for each action-set A_i there exist a P_{max} and P_{min}, which means different sensitive value, which can only be less than or equal to the sensitive value for all the possible packet sizes. Using this new sensitive value that is specific to A_i will affect the noise magnitude leading to less overhead.

Algorithm 10 is an extension to Algorithm 5, where the only difference is for the laplace noise magnitude. In Algorithm 10 we have added the feature where epsilon depends on the sensitive value of packets at this index. for example for the first packet the range is always between[10 : 50] while for the second packet it is between [20 : 25]. This means the sensitive value for the first packet is 40 while for the second packet is only 5. Thus the magnitude of the laplace noise will be less for the second packet as there is not much noise needed to cover the identity of the second packet. We leverage this to extend our previous Algorithm 5 so that we can achieve less overhead without having to sacrifice more utility of our web application.

Example 16 *Given a web application, the maximum packet size that the application can send is equal to 500 while the minimum size is 100, so the sensitive value here for this application is 400. But assuming it only sends a $\vec{a}$ of length 2 for each user, and if for all a in A_1 the minimum and the maximum values for $\vec{v}$ corresponding to that action are 100, = idx)and 200 respectively. This means that the sensitive value at this specific Action-set is 100, same for A_2 the the corresponding $\vec{v}$ can have minimum and maximum size of 400, and 500 respectively which sets the sensitive value at 100 as well.*

This example shows how much the noise magnitude which depends on the sensitivity could be reduced specifically to each different application and in response to each each different action. This can save a lot of overhead.

5.2 Evaluation

We now present our experimental results. We collect testing vector-action sets from real-world web applications, one popular search engine (where users' searching keyword needs to be protected) and one authoritative drug information system from a national institute (where users' possible health information need to be protected). Specially, for the search engine, we collect flow-vectors with respect to query suggestion widget for all possible combinations of four letters. For the drug information system, 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 have to know internal details of the applications. Note that these data are collected using separate programs whose efficiency is not our concern. We compare our algorithms against packet-size rounding, and all experiments are conducted on a PC with 2.20GHz Duo CPU and 4GB memory.

5.2.1 Communication Overhead

In the following experiments, we test the effect of different algorithm parameters, for both grouping and splitting algorithms. For example the effect of delay, packet limit, and sensitivity calibration. Also the effect of the privacy parameter $epsilon$ is considered in the experiment. For different algorithms, we show the average overhead percentage, where the overhead is the extra amount of traffic (padding) that is added to the original word traffic and is calculated using the cost and distance metrics of our privacy model in Section 4.1.3. Through the experiments we consider average word traffic size of "3550 byte" based on the sizes generated from the collected Web applications data. One word corresponds to one user query sent to the search engine application.

The percentage of delayed packets in grouping algorithms is shown in Figure 14. The figure shows also that the percentage varies depending on the different algorithm settings. Furthermore, there is a notable improvement to the grouping algorithm when coupled with packet limit and sensitivity calibration. Packet limit appears to have most of the effect at the lower epsilon values in reducing the percentage of packets that has to be delayed, while the sensitivity calibration is more effective at higher epsilon values. However, the lowest percentage of delayed packets we were able to reach was slightly more than 55%, which means more than half the packets get delayed in the best case scenario of packet grouping. Similarly, the number of delayed packets per word in grouping algorithms has a similar trend to that of the total percentage of delayed packets as shown in Figure 15. Figure **??** shows how the average number of packets per packet group increases with higher epsilon values. Packet limit shows higher number of packets per packet group compared to other algorithm configurations where sensitivity calibration is the most efficient in this case.

Moreover the overhead in the grouping algorithms is compared against the average packet size and the average word size. Consequently, figures 18 and 17 show the trend of overhead percentage as the epsilon value changes. The trend as revealed by the figures, demonstrates the decrease of the overhead. Specifically, at epsilon values higher than 10, different grouping algorithms configurations shows very similar values of overhead. However for epsilon value lower than 10, algorithm configured with sensitivity calibration has

the lowest overhead values.

In packet grouping algorithms, sensitivity calibration shows better results overall. Nonetheless, packet limit helps reduce the percentage of packets delayed, but it results in more overhead at lower epsilon values. Figure 19 illustrates the effect of epsilon value changes to the percentage of delayed packets, the percentage of overhead per average word size and the average number of packets grouped together.

Regarding our designed splitting and merging algorithm explained in Section 4.2.6, plethora of experiments were conducted to compute various aspects and trends of the algorithm. In our experiment, different algorithm configurations were considered along with different values of the privacy parameter *epsilon*, and the packet delay value.

Using different delay values namely, 0, 4, 9, and 14, the percentage of last packet number is computed against epsilon values ranging from 0.1 up to 200, referring to Figure 20. The percentage of last packets decrease significantly with more delay applied, also there is a slight improvement when the value of epsilon is incremented. Regardless that improvement Figure 21 shows that the change in delay value does not have any effect on the size of the last packets. Yet, the increase in epsilon values shows great improvement on the same aspect.

The overhead percentage per word is reduced greately with the increment of the epsilon values, and the increase of delay value shows effictive to lower the overhead even more as shown in Figures 22 and 23. Logarithmic scale is used in the former figure as most of the change happens at the smaller epsilon values. Moreover, Figures 24 and 25 illustrates the general behaviour of packet splitting and merging algorithm regarding different aspects such as the percentage of the overhead, number and size of the last packets.

Furthermore, Figures 26, 27 and 28 show how different splitting and merging algorithm configurations can affect the outcome of the algorithm. The experiment are computed with fixed packet delay value of 2 to enable us to manipulate other parameters. For instance the relation between the overhead amount and the epsilon change is shown in Figure 28. The figure shows also that sensitivity calibration with packet limit applied lead to lower overhead values. It also is shown that the same configuration leads to lower last packet

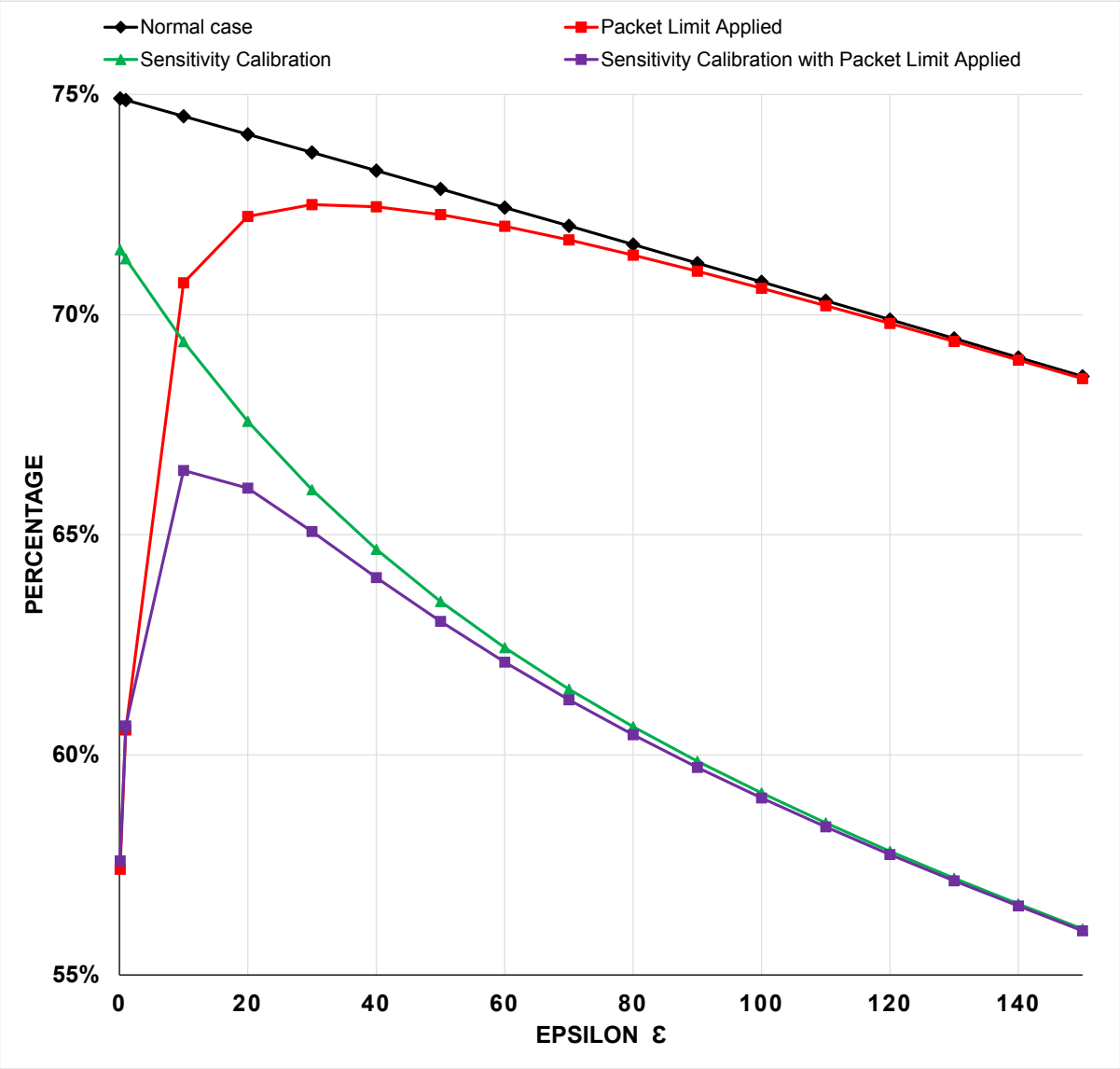

Figure 14: Percentage of delayed packets in Grouping Algorithms

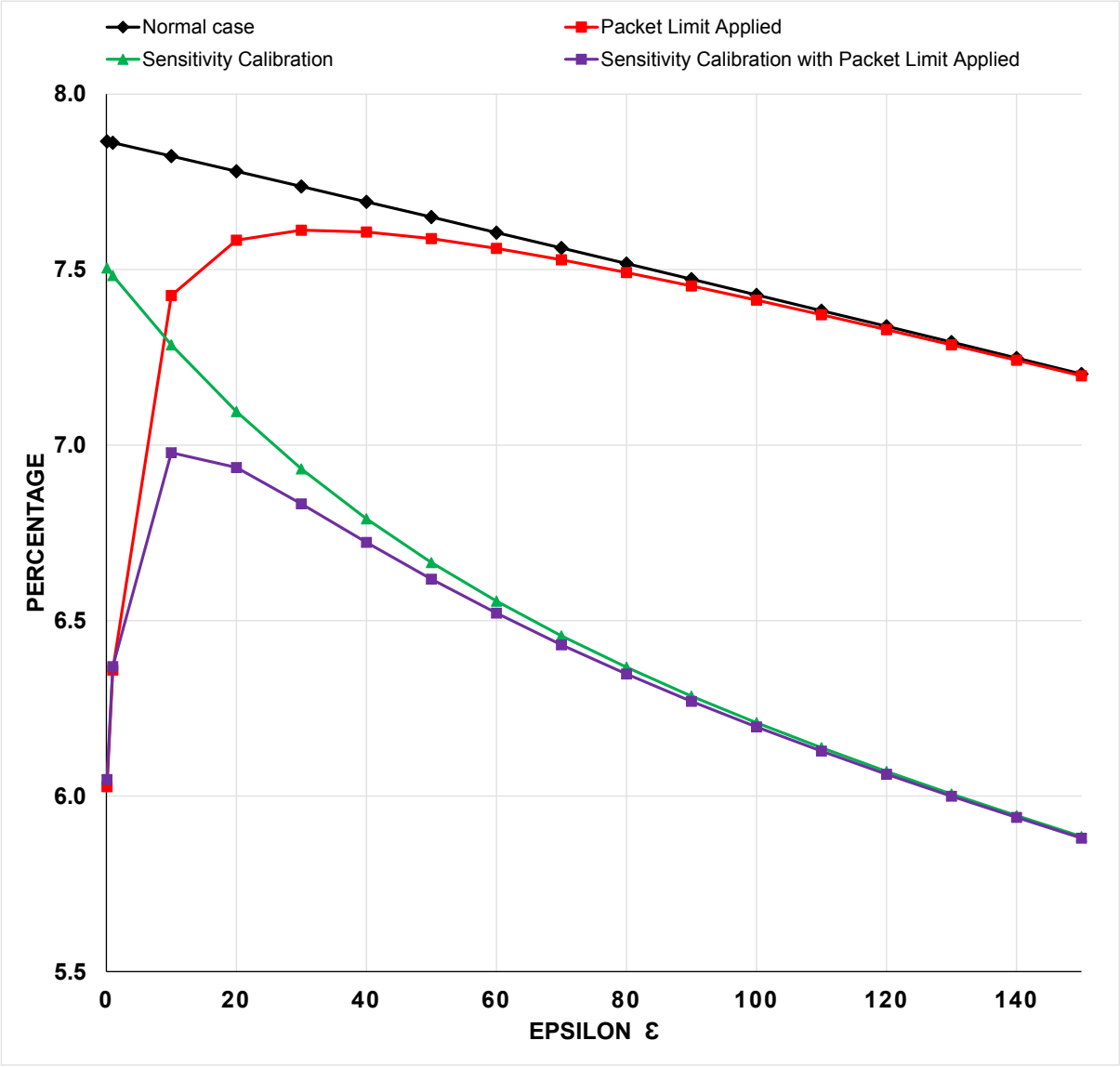

Figure 15: Average number of delayed packets per word in Grouping Algorithms

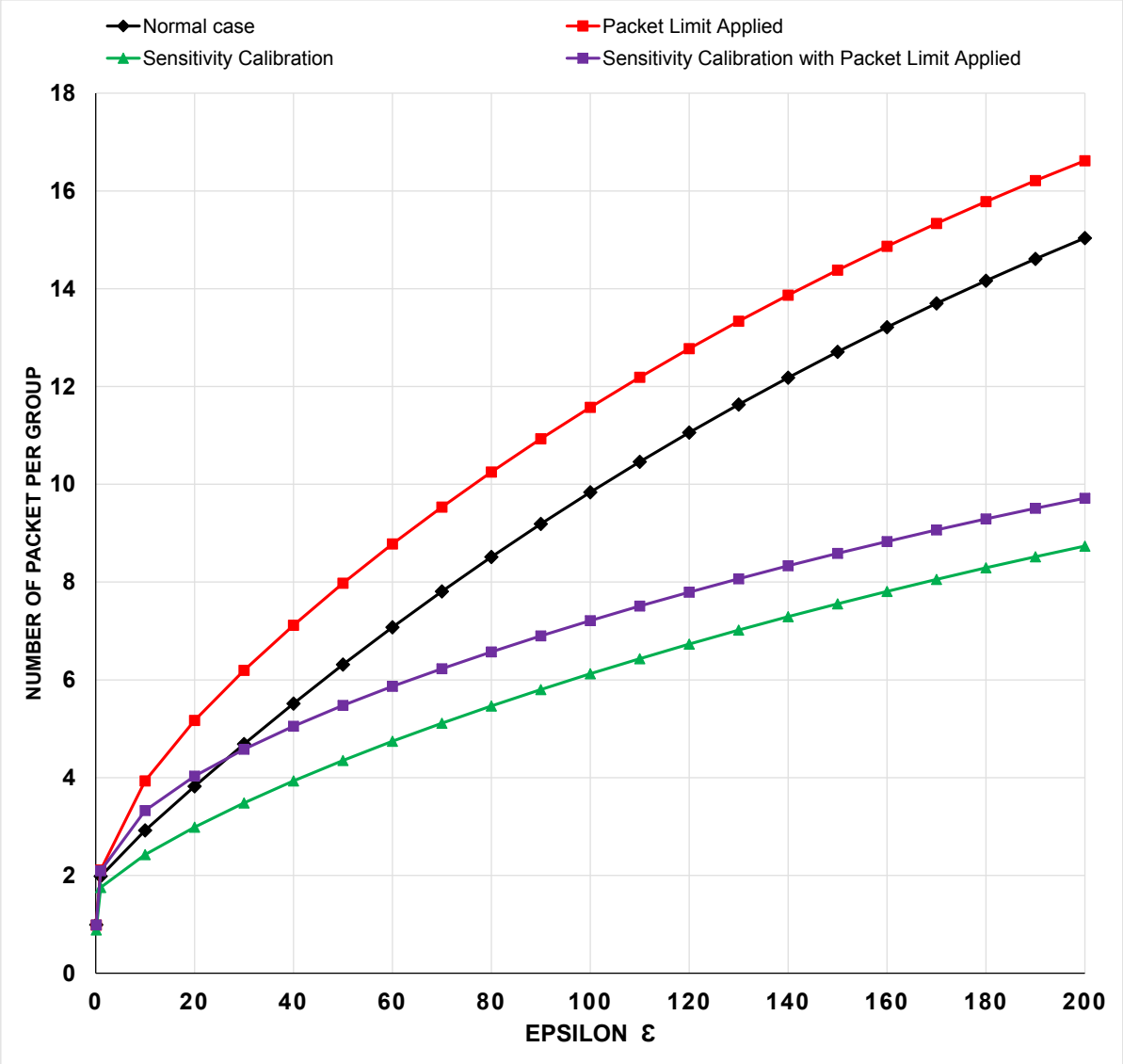

Figure 16: Average number of packets within the same group in Grouping Algorithms

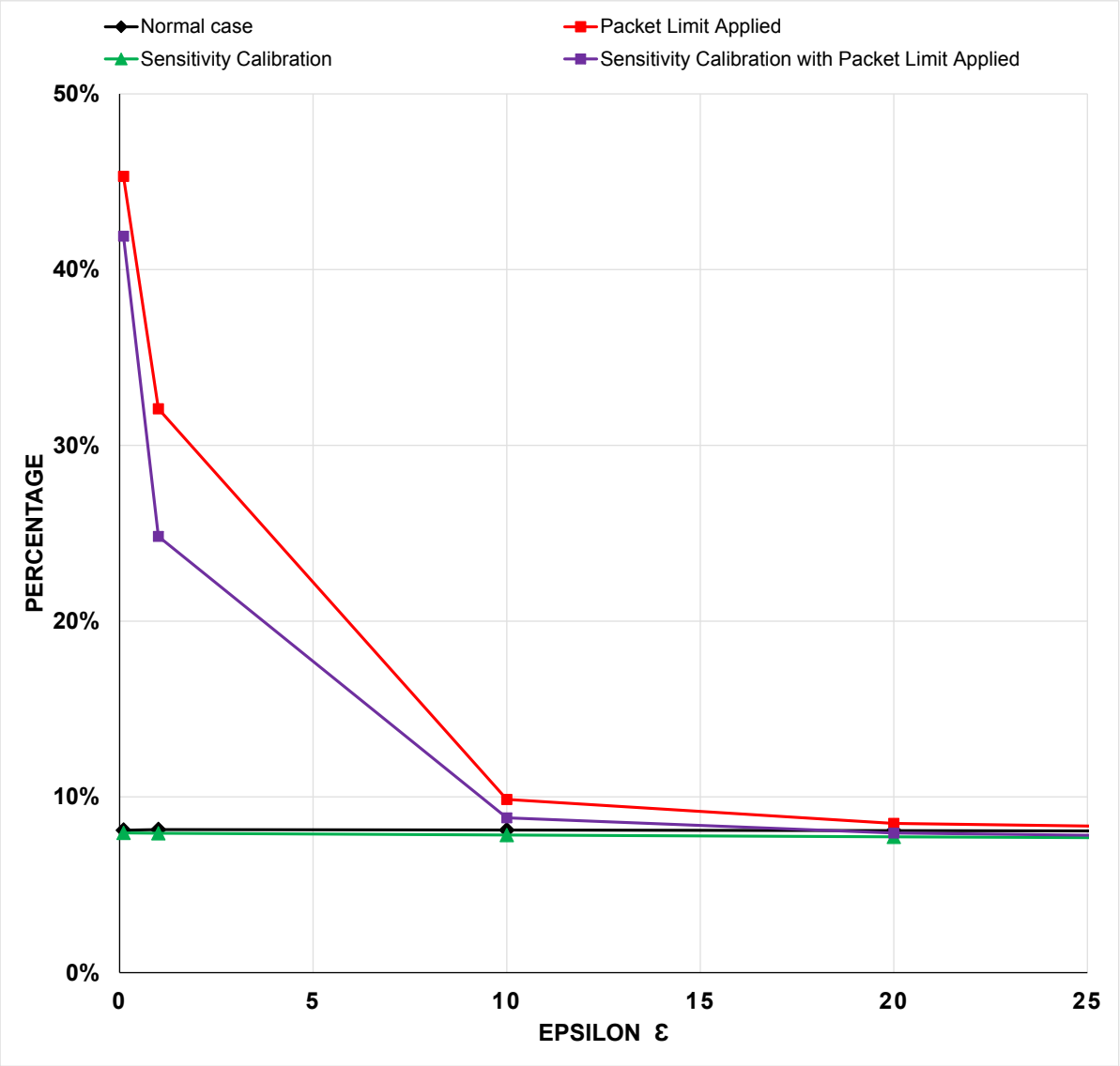

Figure 17: Percentage of overhead per word in Grouping Algorithms

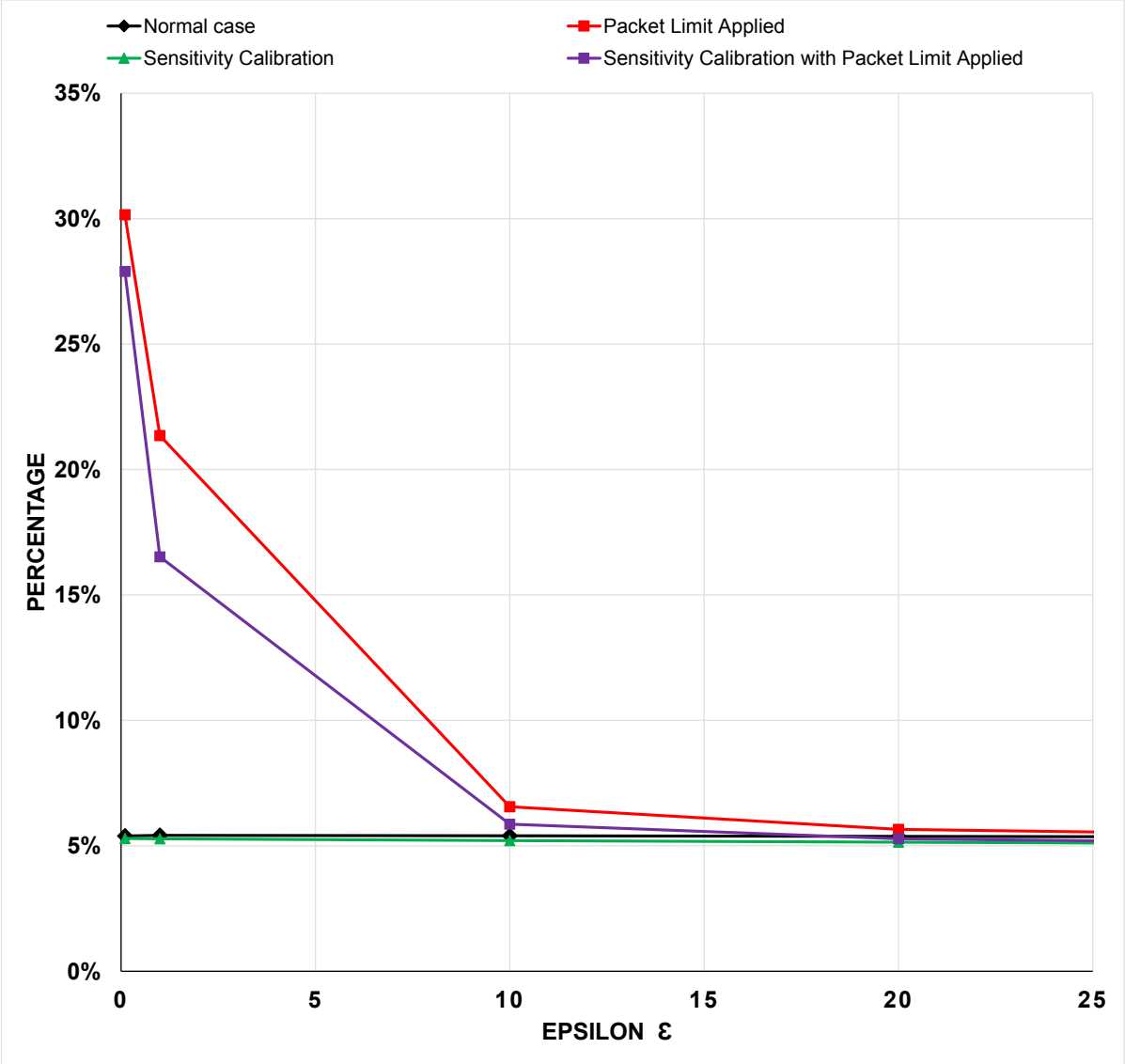

Figure 18: Percentage of overhead per packet in Grouping Algorithms

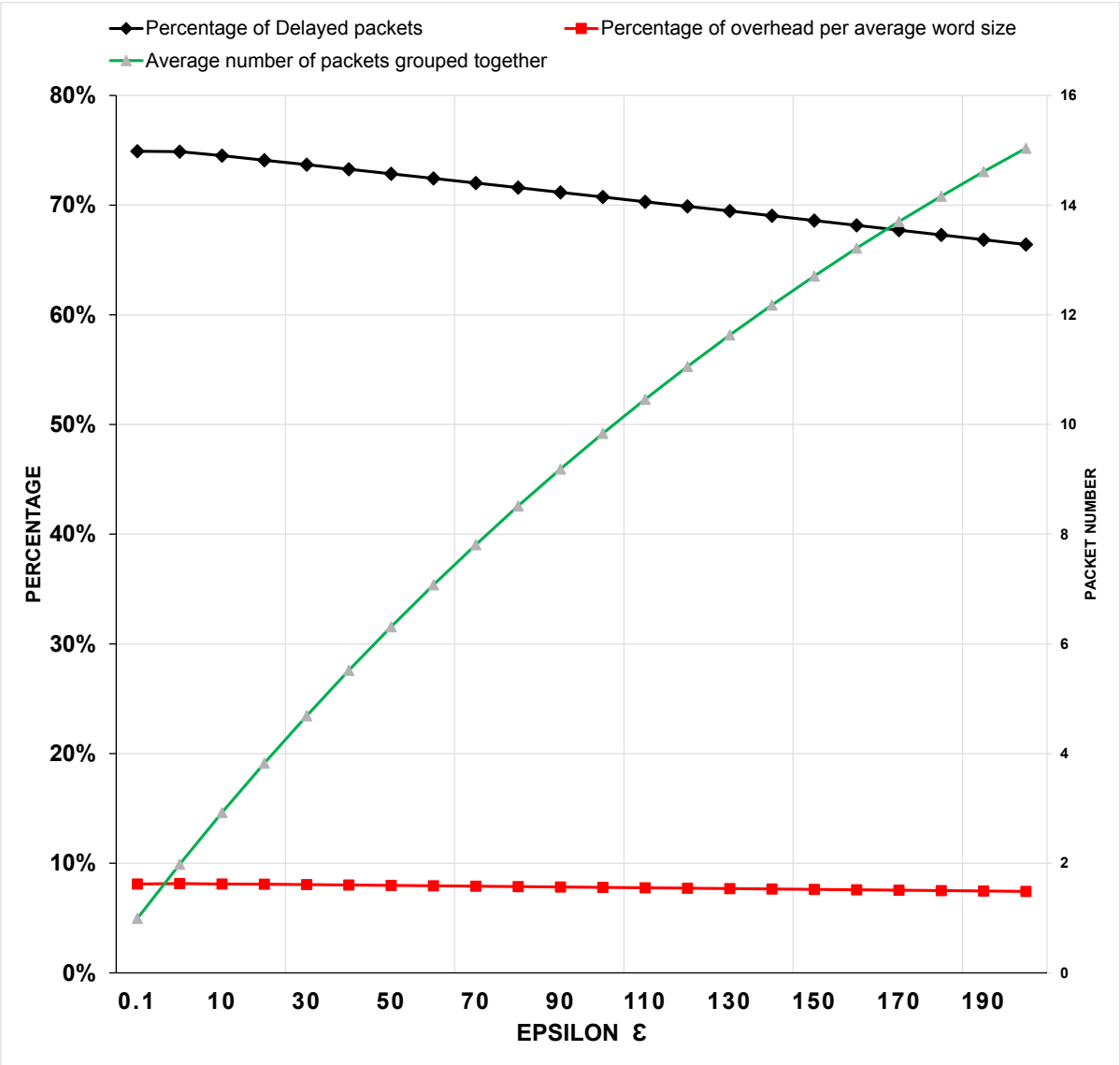

Figure 19: Trends in Grouping Algorithms

numbers and sizes as shown in Figure 26 and 27, respectively.

Finally, our experimental results provides the strong evidence that our designed splitting and merging algorithm with its various configuration provides better overhead reduction for the problem of traffic padding compared to other existing solutions. Also, the novel adoption of differential privacy in our model enables our algorithms to provide the required privacy guarantees. After all, our solution provides as well the flexibility and the adaptability of to the different application needs, as we proposed different configurations and parameters that can be tailored to target the diverse utility needs.

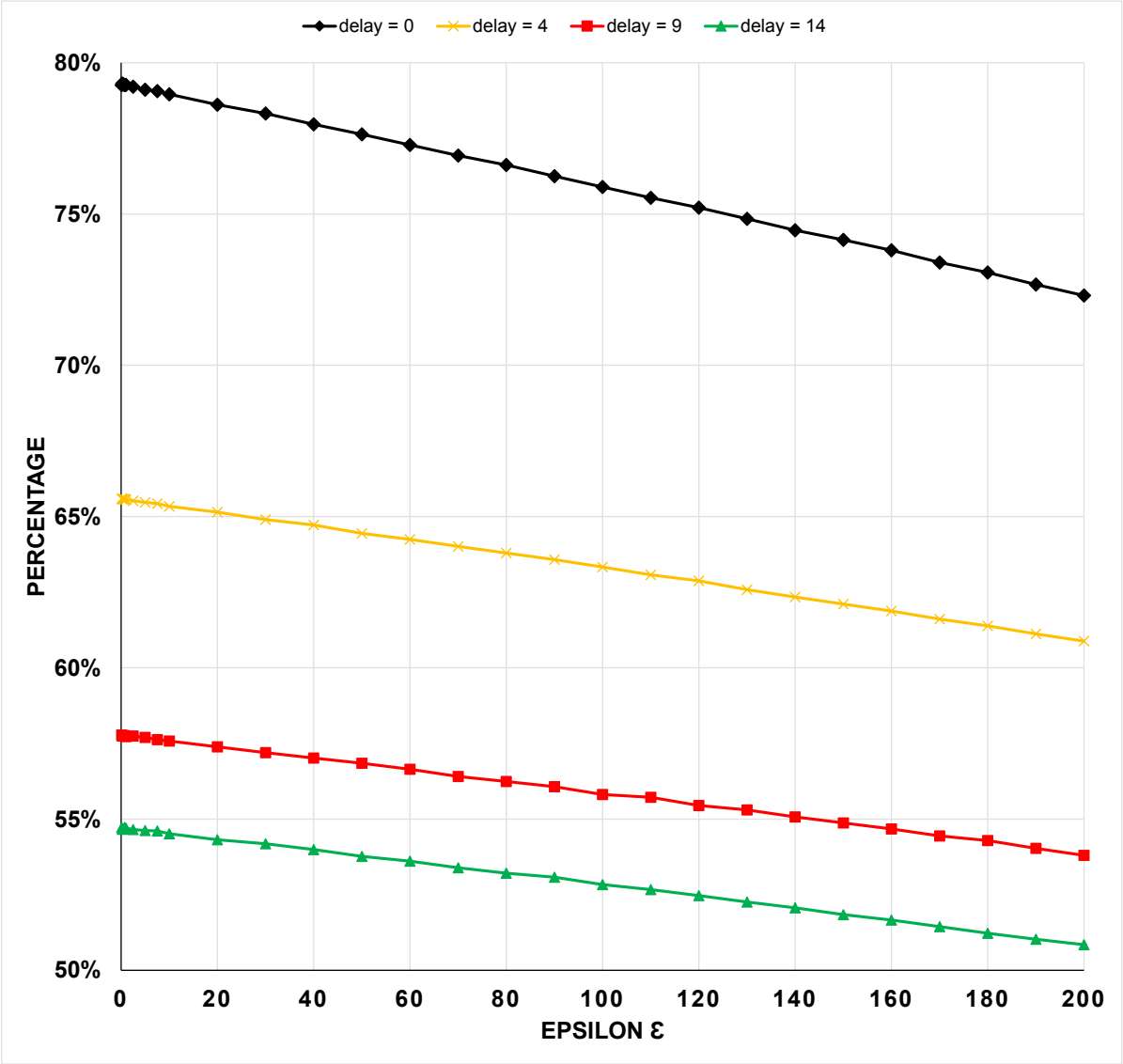

Figure 20: Percentage of last packet number

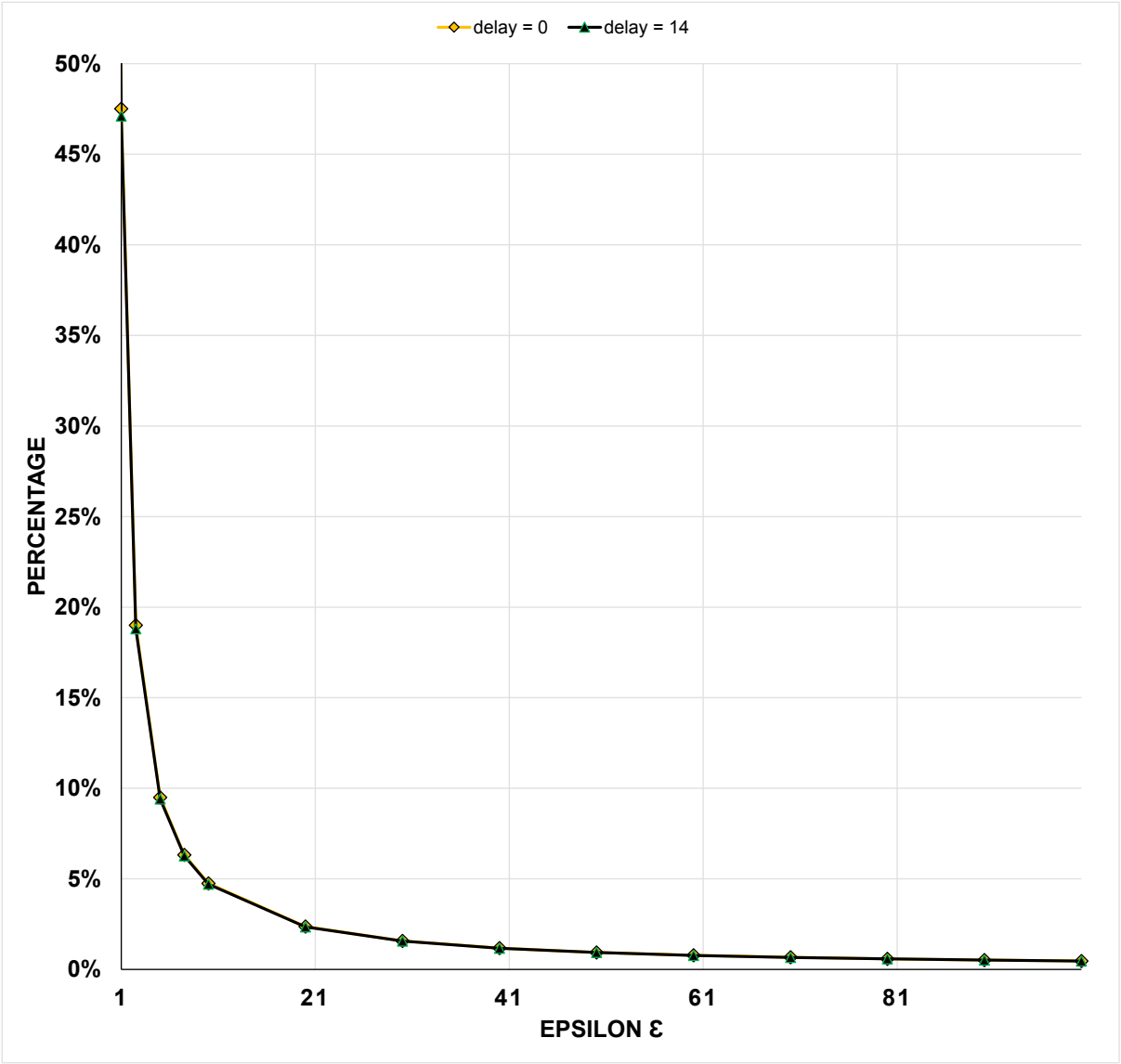

Figure 21: Percentage of last packet size per average word size

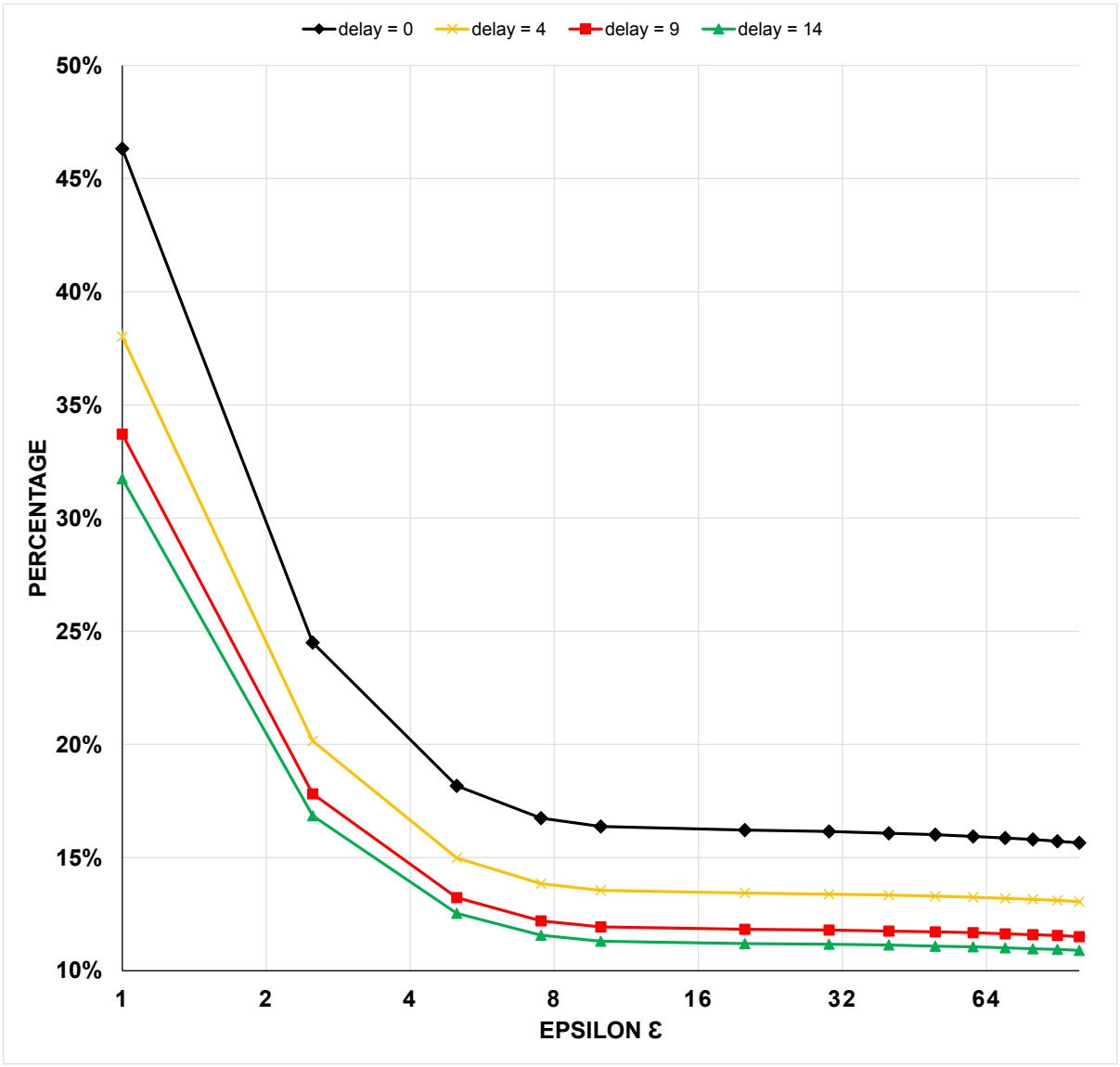

Figure 22: Percentage of overhead with fixed delay value

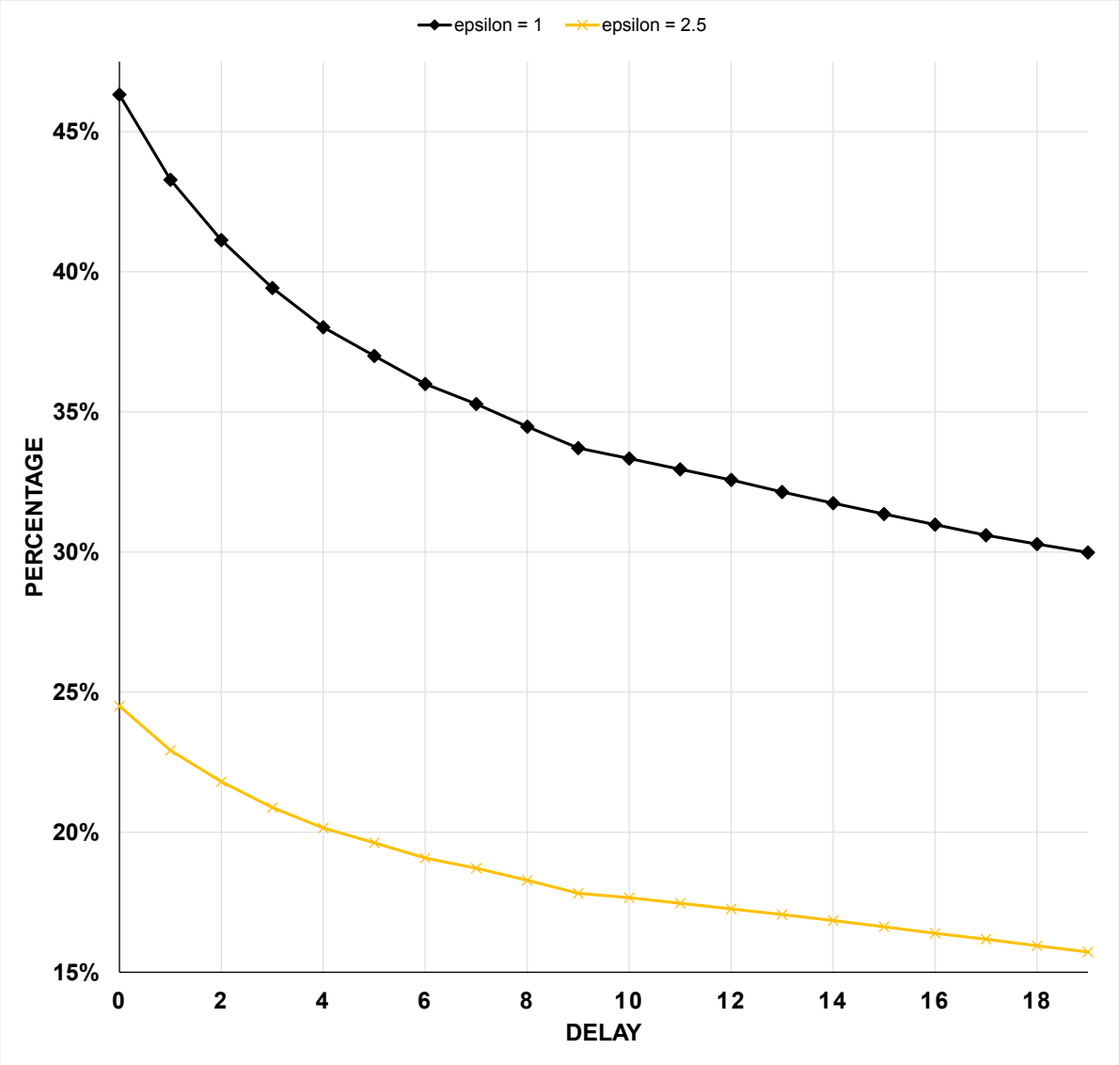

Figure 23: Percentage of overhead with fixed epsilon

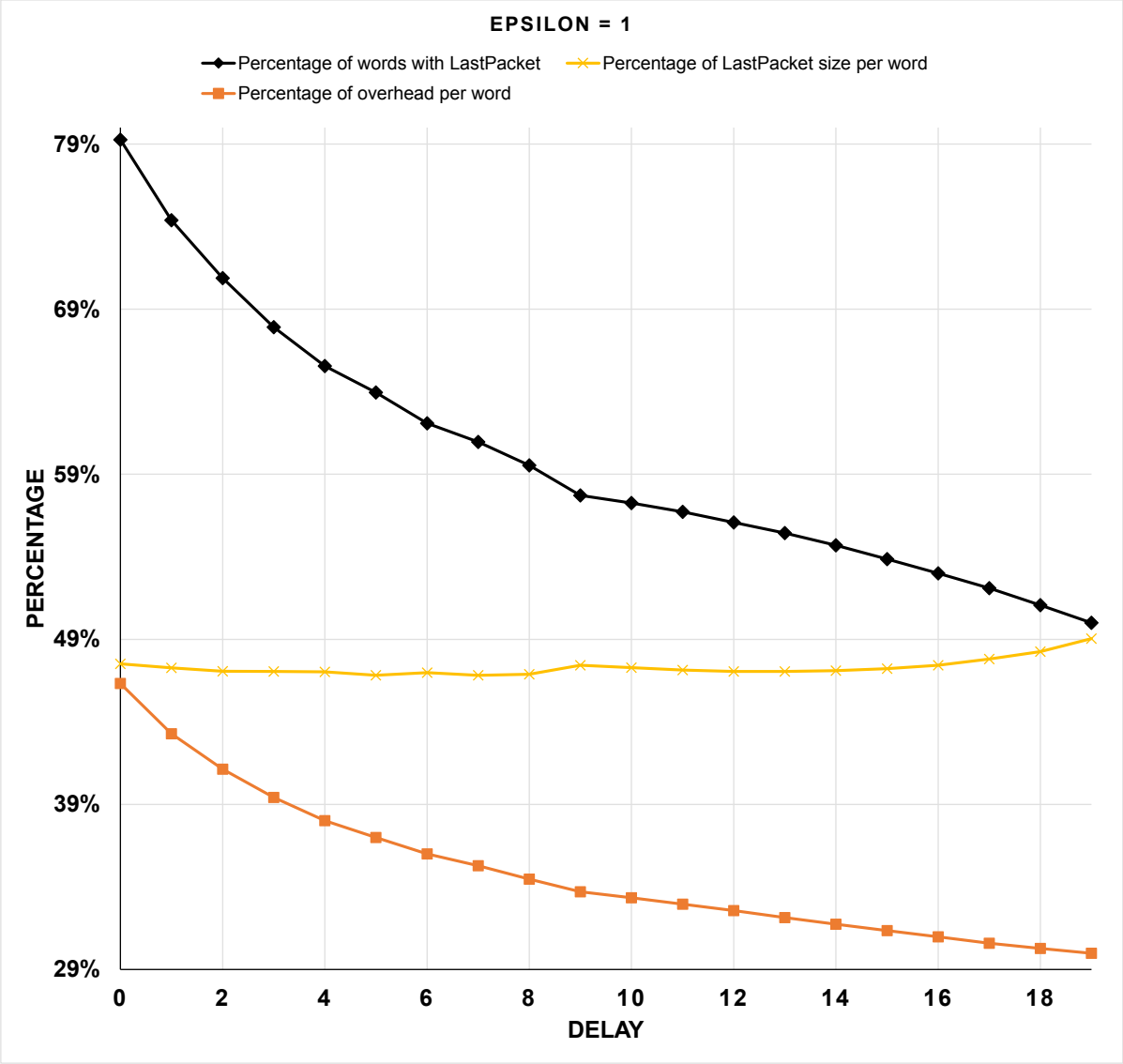

Figure 24: Trend in splitting and merging algorithms over delay value

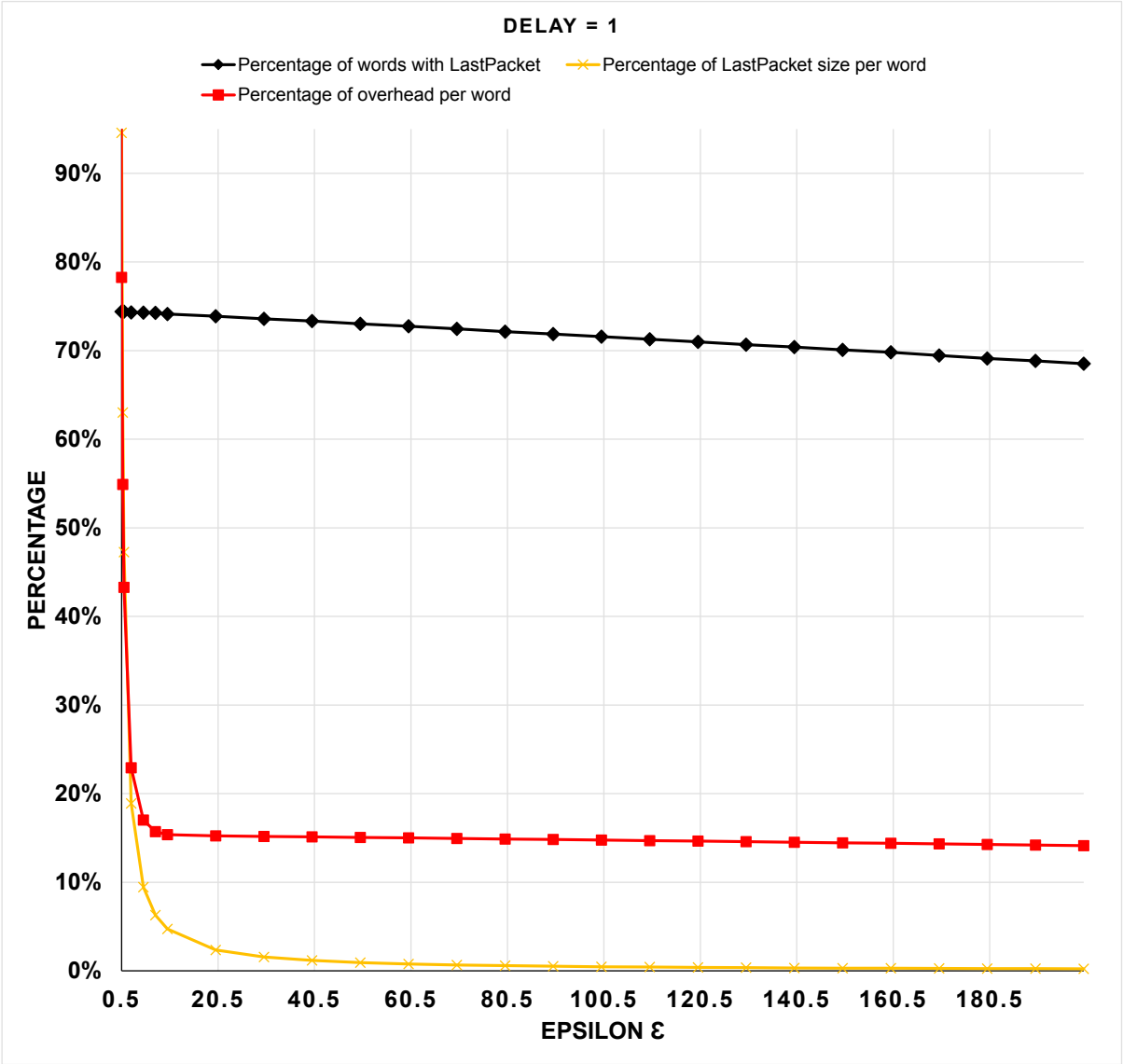

Figure 25: Trend in splitting and merging algorithms over epsilon value

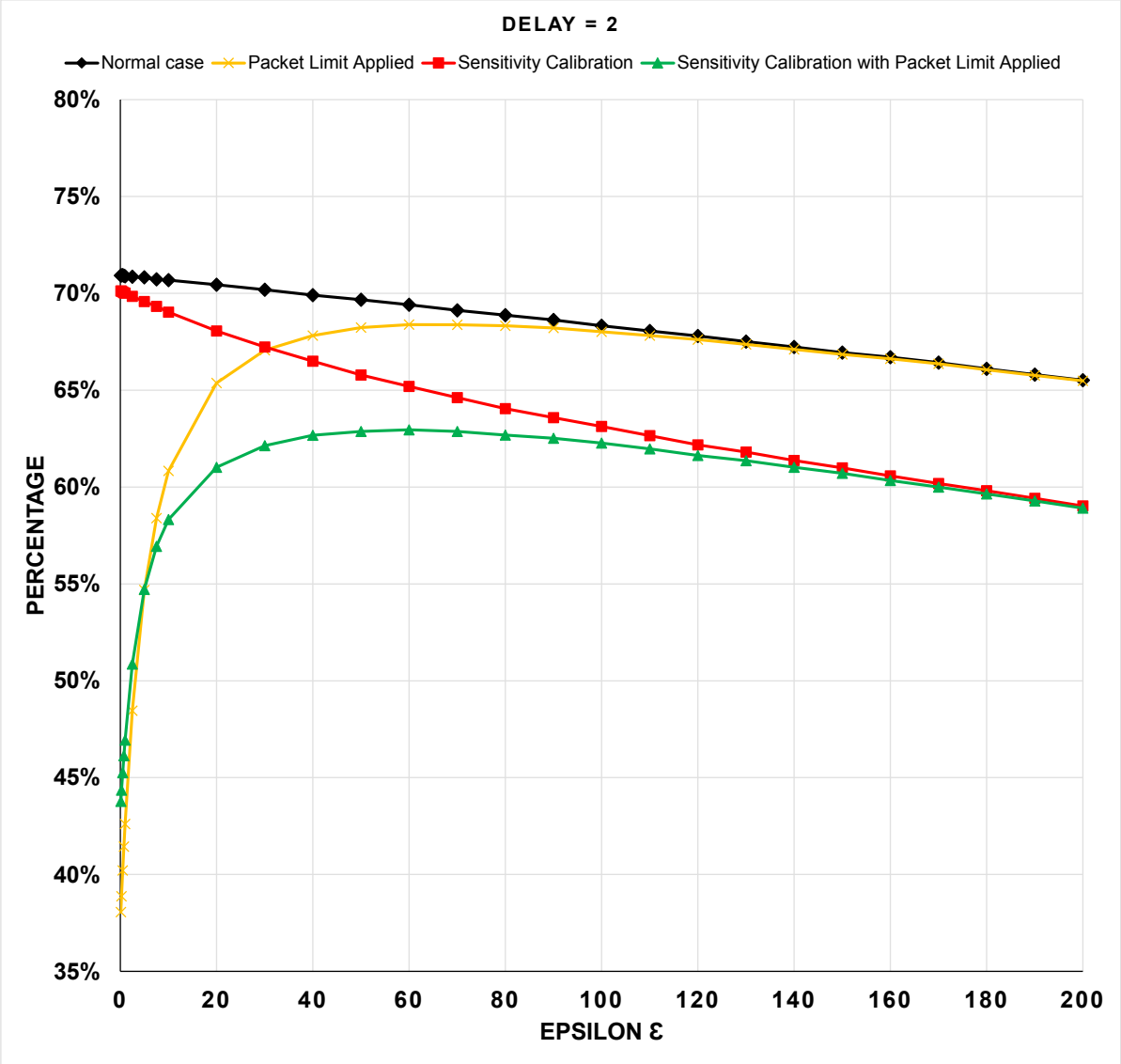

Figure 26: Percentage of last packets for different configurations

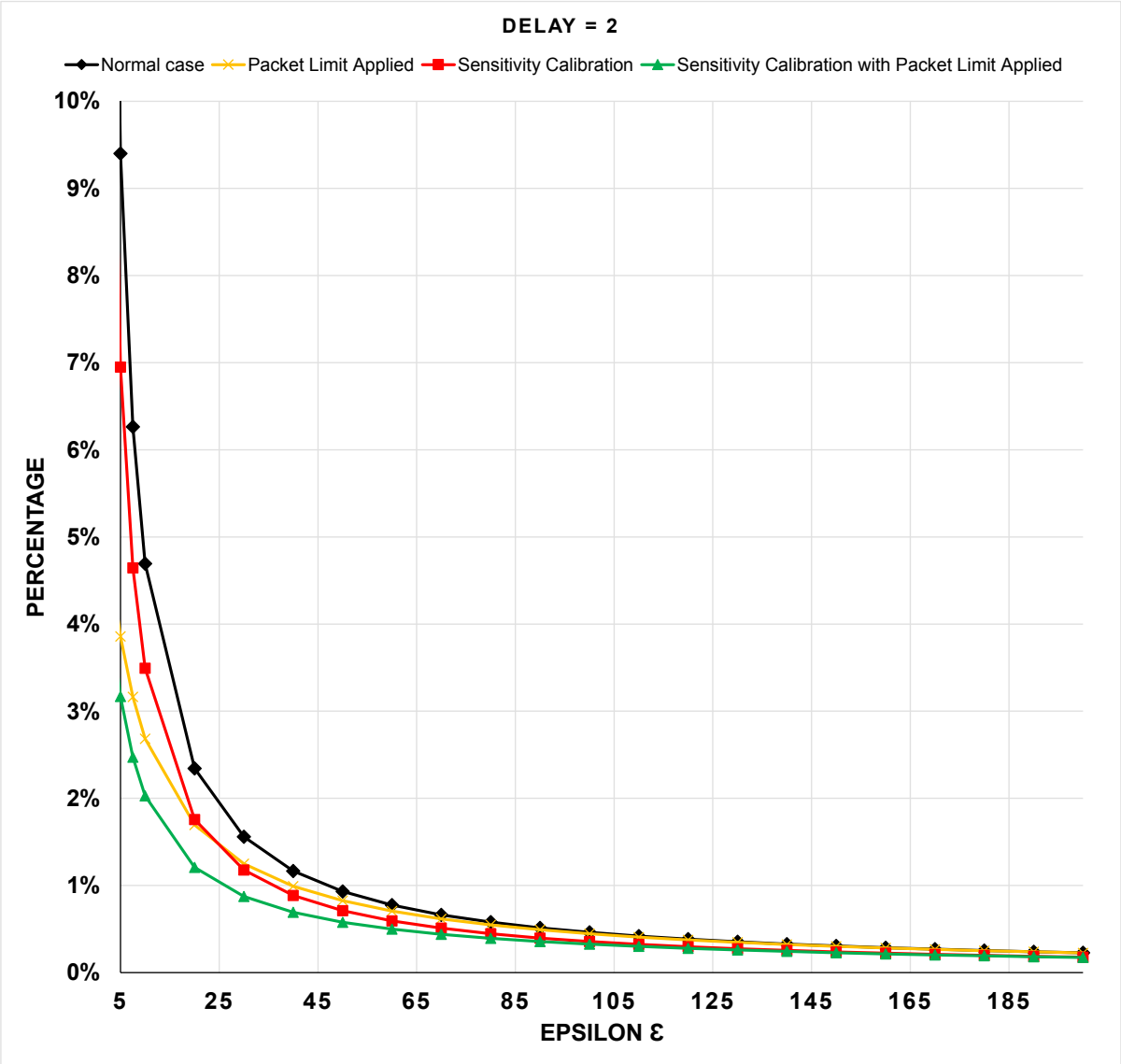

Figure 27: Percentage of last packet size per average word size for different configurations

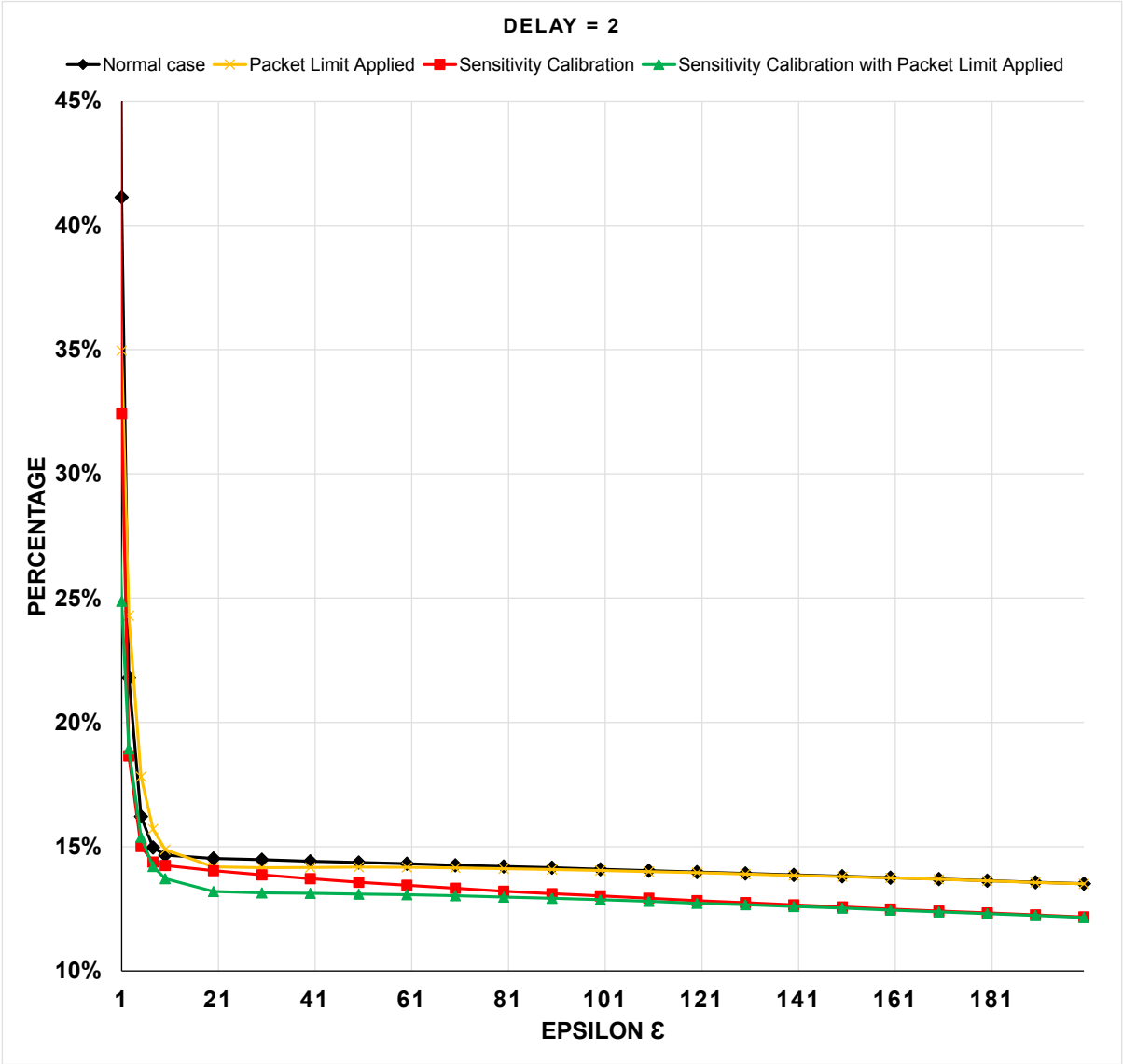

Figure 28: Percentage of overhead for different configurations

Algorithm 10: Packet Splitting and Merging with Sensitivity Calibration - Part 1

```
Data: List packets to be send P_1....P_m;
// C = the carry from previous packet.;
// S = size to be send;
// index is the index of the packet processed.
C = null;
foreach packet P do
    n = generateSensitivityCalibratedNoise(index);
    P_x = concatenate(C, P);
    //1- determine the size to be send
    S = size(P) + n;
    //2- prepare the data to be send
    if size(P_x) < S then
        while size(P_x) < S do
            pad(P_x);
        end
    else
        if size(P_x) == S then
            send(P_x);
        else
            // size(P_x) >= S;
            new Packet P_a = subpacket(P_x, S);
            C = P_x - P_a;      //will be process with the next packet
            send(P_a);
        end
    end
end
```

Algorithm 11: Packet Splitting and Merging with Sensitivity Calibration - Part 2

```
1 while size(C) >= S_max do
2     new Packet P_a = subpacket(C, S_max);
3     C = C - P_a;
4     send(P_a);
5 end
6 if size(C) > 0 then
7     pad(C, S_max);
8     Send(C);
9 end
```

Chapter 6

Conclusion

In this thesis, we concentrated on the study of differentially private traffic padding. Throughout the thesis,

- We have demonstrated the effectiveness of side channel attacks on encrypted traffic between users and Web applications.
- We have provided a first effort on modeling the privacy requirements using the differential privacy concept.
- We have established a mapping between the traffic padding problem and the existing differential privacy model.
- We provided a series of concrete algorithms to address the issue of negative noise sizes, and other practical issues, such as the last packet challenge, in order to improve the performance while ensuring differential privacy.
- We have evaluated the performance of our proposed algorithms using data collected from real world applications.

Our future work will be directed toward the implementation of the proposed server-side padding mechanisms using open source Web applications, by which we will be able to better evaluate the server-side performance and the practicality of our solution, and to

investigate the best approaches that simplify the maintenance of changing requirements and application design.

Bibliography

[1] Part 11:IEEE 802.11-2007, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, June 2007., June 2007.

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

[3] A. Askarov, D. Zhang, and A.C. Myers. Predictive black-box mitigation of timing channels. In *CCS '10*, pages 297–307, 2010.

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

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

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

[7] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In *USENIX Security Symposium*, pages 307–322, 2010.

[8] 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.

[9] 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.

[10] Andrea Bittau, Mark Handley, and Joshua Lackey. The final nail in wep's coffin. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

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

[12] 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.

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

[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] Valentina Ciriani, S De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati. k-anonymous data mining: A survey. In *Privacy-preserving data mining*, pages 105–136. Springer, 2008.

[16] Jim Conallen. Modeling web application architectures with uml. *Communications of the ACM*, 42(10):63–70, 1999.

[17] 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.

[18] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210. ACM, 2003.

[19] Cynthia Dwork. Differential privacy. In *in ICALP*, pages 1–12. Springer, 2006.

[20] Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[21] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, pages 265–284. Springer, 2006.

[22] Cynthia Dwork and Kobbi Nissim. Privacy-preserving datamining on vertically partitioned databases. In *Advances in Cryptology–CRYPTO 2004*, pages 528–544. Springer, 2004.

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

[24] 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.

[25] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. Cryptology ePrint Archive, Report 2013/857, 2013.

[26] X. Gong, N. Kiyavash, and N. Borisov. Fingerprinting websites using remote traffic analysis. In *CCS '10*, pages 684–686, 2010.

[27] Ori Heffetz and Katrina Ligett. Privacy and data-based research. Technical report, National Bureau of Economic Research, 2013.

[28] Xiaoqian Jiang, Zhanglong Ji, Shuang Wang, Noman Mohammed, Samuel Cheng, and Lucila Ohno-Machado. Differential-private data publishing through component analysis. *algorithms*, 13:14.

[29] Aleksandra Korolova. Privacy violations using microtargeted ads: A case study. In *ICDM Workshops*, pages 474–482, 2010.

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

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

[32] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 255–263. ACM, 2006.

[33] Wen Ming Liu, Lingyu Wang, Kui Ren, Pengsu Cheng, and Mourad Debbabi. k-indistinguishable traffic padding in web applications. In *Privacy Enhancing Technologies*, pages 79–99. Springer, 2012.

[34] Wen Ming Liu, Lingyu Wang, Kui Ren, and Mourad Debbabi. Background knowledge-resistant traffic padding for preserving user privacy in web-based applications. Technical report, Technical report, Concordia University, 2013. Available at http://users. encs. concordia. ca/ l wenmin/cloudcom2013 long. pdf.

[35] 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*.

[36] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity.

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

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

[39] 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.

[40] Aaron Roth. New Algorithms for Preserving Differential Privacy, 2010.

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

[42] 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.

[43] 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.

[44] 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.

[45] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy*. Society Press, 2002.

[46] Qixiang Sun, Daniel R Simon, Yi-Min Wang, Wilf Russell, Venkata N Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 19–30. IEEE, 2002.

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

[48] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In *INTERNATIONAL WORKSHOP ON DESIGNING PRIVACY ENHANCING TECHNOLOGIES: DESIGN ISSUES IN ANONYMITY AND UNOBSERVABILITY*, pages 96–114. Springer-Verlag New York, Inc., 2001.

[49] Lilla Tthmrsz. *Differential Privacy* . PhD thesis, Etvs Lornd University ,Faculty of Natural Sciences.

[50] 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.

[51] Charles V. Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 35–49, Washington, DC, USA, 2008. IEEE Computer Society.

[52] 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.

[53] Li Zhuang, Feng Zhou, and J Doug Tygar. Keyboard acoustic emanations revisited. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 373–382. ACM, 2005.