HIDING BEHIND THE CLOUDS: EFFICIENT, PRIVACY-PRESERVING QUERIES VIA CLOUD PROXIES

Surabhi Gaur
San Jose State University

HIDING BEHIND THE CLOUDS:

EFFICIENT, PRIVACY-PRESERVING QUERIES VIA CLOUD PROXIES

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Surabhi Gaur

May 7, 2014

The Designated Project Committee Approves the Writing Project Titled

HIDING BEHIND THE CLOUDS:

EFFICIENT, PRIVACY-PRESERVING QUERIES VIA CLOUD

PROXIES

by

Surabhi Gaur

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2014

Dr. Melody Moh	Department of Computer Science
Dr. Teng-Sheng Moh	Department of Computer Science
Rafael Alvarez-Horine	Department of Computer Science

Abstract

This project proposes PriView, a privacy-preserving technique for querying third-party services from mobile devices. Classical private information retrieval (PIR) schemes are difficult to deploy and use, since they require the target service to be replicated and modified. To avoid this problem, PriView utilizes a novel, proxy-mediated form of PIR, in which the client device fetches XORs of dummy query responses from each of two proxies and combines them to produce the required result. Unlike conventional PIR, PriView does not require the third-party service to be replicated or modified in any way. We evaluated a PriView implementation for the Google Static Maps service utilizing an Android OS front-end and Amazon EC2 proxies. PriView is able to provide tunable confidentiality with low overhead, allowing bandwidth usage, power consumption, and end-to-end latency to scale sublinearly with the provided degree of confidentiality.

Acknowledgements

This Masters thesis is the result of the help and support of people in my professional and personal life, without whom, my efforts would not have come to fruition. First and foremost of them would be my advisor, Dr. Melody Moh, who believed in me when I didn't and who pursuaded me to trust my capabilities. I have tested her kindness and patience on innumerable occasions, only to realize that she has immense reserves of them. Her guidance and support made this project reach its potential and result into posters, a conference paper and finally, a Masters thesis. She has been a mentor and a guide and an advisor in the true sense of the word and I am extremely grateful to her.

I would also like to express my sincere thanks to my committee members - Dr. Teng-Sheng Moh: thank you very much for making my defense an enjoyable experience. I value your comments and suggestions very much; Rafael Alvarez-Horine: thanks for having done all this right before me and for sharing your experiences and valuable wisdom, I escaped many a pitfalls because of them. Not to mention the immense knowledge of Cloud Computing that was such a huge help with the basic premise of the project.

This acknowledgement would be incomplete without thanking Dr. Sami Khuri - he converted me to a CS major by just one course of his. He is one of the few incredibly inspiring people I have had the pleasure to meet in my life. The other people that came into my life through this Masters and made it an incredibly fun experience, were the friends I made here. They helped me grow by their knowledge and drive, they helped me through difficult times and made every minute I spent on campus a very happy experience.

The kind of support and incouragement I get from the people in my workplace, Hewlett-

Packard, is beyond belief. They have played an important part in my being able to complete this degree and I am extremely grateful to them.

On the personal front, my parents and brother have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice. My in-laws have been pillars of support and have encouraged me endlessly - I can't thank them enough for being there for me always. I would also save a tiny, little thanks for my tiny, little furbaby, Dunkie. Her warm cuddles and unconditional love got me through many all-nighters.

Above all, I would like to acknowledge the amazing support of my partner-in-crime and the reason behind my ever stepping into this degree, Mahesh. He gave me a new life, a new mindset and a new career and more importantly, he learned how to make endless cups of *chai* to get me through this Masters. I am not quite sure what I did to deserve his love and the kindness of all the people that helped me in every possible way through the last three years, but I truly cherish every memory and I am thankful for the incredible experince this Masters has been.

Contents

List of Figures

1 Introduction

The emergence of ubiquitous, high-bandwidth 3G/4G connectivity has enabled a new class of Internet services designed to be accessed from smartphones and tablets. Examples range from mundane services such as maps and restaurant reviews to more exotic applications such as augmented reality (e.g., the Google Glass [1]). Unfortunately, the convenience of ubiquitous connectivity comes at the price of privacy. When a user accesses a map of her immediate surroundings, she reveals her location to the mapping service. When she looks up the price of a house or a car, she provides information to marketers. When she browses restaurant reviews, she reveals her dietary preferences to the review website.

1.1 Existing solutions for privacy-preservation

Existing solutions protect privacy by sacrificing either efficiency or accuracy. In the first category are solutions where the user's device makes multiple 'dummy' queries to the Internet service for each valid query [2]. For example, a user accessing a mapping service might provide multiple locations in order to hide her actual location; the service then knows that she is at one of the locations but does not know which one. However, this approach results in multiple responses (in this example, maps) being returned to the device from the service, wasting both bandwidth and power on the device. A solution of the second category involves 'jittering' the input [3], preventing the service from knowing the exact input (e.g., the exact location of the user); however, this means that the user is returned approximate answers (e.g., a map of a nearby location), which may not always be useful. A different type of solution places a cloud-hosted proxy between the user and the service; however, this requires the user to completely trust the cloud provider, who is aware of both the identity of the user as well as the contents of the query.

1.2 Introduction to PriView

In this paper, we present PriView, a new technique that allows mobile devices to access Internet services without sacrificing privacy, efficiency or accuracy. In Priview, to send a query to an Internet service, the device includes it along with a single set of dummy queries that it sends to two different cloud-hosted proxies. Each cloud proxy relays the set of queries to the third-party service and sends back an XOR of a predetermined subset of the responses to the device. The subsets are chosen *a priori* by the client such that they are identical across the two proxies, except that one contains the actual query's response while the other does not. This ensures that combining the two XORs cancels out dummy responses and provides the response to the actual query.

The PriView technique preserves data confidentiality. When a user issues a query using it, no entity in the system can determine whether it's a real or dummy query. Each cloud proxy is aware of the identity of the querying device but does not know the actual query being issued, since it is provided with a set of dummy queries. The Internet service knows neither the exact query being issued nor the identity of the querying device. Unless the two cloud proxies collude with each other, no single entity is aware of the exact query that was issued. In practice, our assumption that cloud proxies do not collude can be satisfied by running each proxy in a different cloud provider (e.g., on Amazon EC2 [4] and Microsoft Azure [5]).

Importantly, PriView achieves these privacy properties efficiently. For each query, the device sends two outgoing messages (one to each proxy), each of which is a list of dummy query inputs that fits into a single network packet. It receives back two XORs, each of which is the size of a regular response. Accordingly, the bandwidth and power consumption of our technique scale sub-linearly with the degree of confidentiality (i.e., the size of the dummy set), in contrast to the linear scaling provided by conventional dummy queries.

Increasing the size of the dummy set – and hence, increasing the degree of confidentiality – has a sub-linear impact on bandwidth and power consumption. The number of messages sent and received do not change (until the dummy set no longer fits in a single outgoing network packet). In other words, changing the degree of confidentiality from 8 to 16 increases the bandwidth and power consumption by much less than a factor of two. As a result, a PriView query uses approximately twice the bandwidth of a conventional query, and uses up a correspondingly larger amount of battery life. In addition, each query requires extra computation in the form of the XOR operation to retrieve the actual response, which results in increased power consumption. Unlike existing schemes based on dummy inputs, increasing the size of the dummy set – and hence, increasing the degree of confidentiality – has an insignificant impact on bandwidth or power consumption, since the number of messages sent and received do not change (until the dummy set no longer fits in a single outgoing network packet).

PriView can be viewed as an adaptation of classical Private Information Retrieval (PIR) schemes [6]. Such schemes typically require the service (or database) to be replicated multiple times, as well as modified to return compact summaries (such as XORs) of multiple items. Replication or modification is typically not possible with real-world Internet services. Instead, PriView inserts a replicated proxy layer between the end user and the Internet service, and then uses PIR between the client and the proxies. This allows PriView to implement a PIR-like scheme against an unmodified, unreplicated third-party service such as Wikipedia or Google Static Maps [7].

We implemented a PriView client for Google Static Maps on the Android platform, with cloud proxies running Amazon EC2. In our evaluation on an Android device, we compare PriView against the strawman solution of issuing dummy queries. For a degree of confidentiality equivalent to that obtained by issuing 64 dummy queries in the strawman solution, PriView uses 14% of the bandwidth and 16% of the power consumed by the strawman, while delivering responses at 40% of the latency.

The rest of this report is organized as follows. In section 2 we discuss the existing work in the field of user/query privacy. Section 3 explains the PriView system in detail, including setup of the PIR scheme and implementation of the model proposed. A performance evaluation is conducted in section 4 and section 5 concludes the paper.

2 Background

Our primary goal is the following: when a user Bob sends a query Q to an Internet service S using a mobile device, our goal is to guarantee that the service S cannot determine that "Bob searched for Q". Further, we want to provide this guarantee without adding too much overhead on the client device in terms of bandwidth usage, end-to-end latency, or power consumption. Finally, we want to make minimal assumptions about the trustworthiness of third party entities such as cloud providers.

Existing solutions can be categorized according to the privacy guarantees they provide:

2.1 Anonymity

One option is to prevent service S from knowing who issued the query. This can be done by inserting an anonymizing proxy between the device and the service. If a single proxy is used, this approach has the disadvantage that the proxy can now see both the query as well as the source IP address. End-to-end encryption at the application level between the client device and the service S eliminates this problem; now the proxy can only see who sent the request, while the service S can only see the contents of the request. However, traffic analysis can still tease out the relationship between the request sent by the device and that received by the service S; this problem can be solved by using more sophisticated proxying techniques such as Onion Routing that use a network of proxies between the client and the server.

Unfortunately, while techniques such as Onion Routing can effectively hide the network address of the user's device, the service can still discover the identity of the user's device from semantic information in the query itself [3]. For example, if the user searches for "Thai restaurants near 42 Marlin Drive", the service can infer that the person who lives at 42 Marlin Drive likes Thai food, and use public databases to identify this person as Bob.

2.2 k-Anonymity

Accordingly, a second class of solutions attempts to prevent the service S from knowing who exactly out of a group of k people issued the query (a guarantee known as k-anonymity [8]). In the case of location-based services, this can be achieved via 'spatial cloaking', where the location included in the query is jittered slightly so that any person within a particular radius of a location could have issued it. For example, the query might become "Thai restaurants near 46 Marlin Drive", in which case the service S does not know which resident of Marlin Drive actually issued the query. However, this solution only works for applications where the input domain is continuous and can be jittered without rendering the output unusable; for example, if Bob wants to search for "the history of Thailand" on Wikipedia, it's not clear how we would jitter this input, and whether the resulting output would still be useful to Bob. As a result, spatial cloaking is useful mostly in the context of location-based applications where certain population density conditions are met, and is too specific for our use case of querying general-purpose Internet services.

A different way to achieve k-anonymity is to generate a set of dummy queries in addition to its original query; for example, the client might send the queries "Thai restaurants near 42 Marlin Drive", "Thai restaurants near 77 Turtle Ave", and "Thai restaurants near 24 Swordfish Road", in which case the service cannot determine which of the three users issued the query. One problematic aspect of this approach is that it requires a trusted anonymizing proxy that knows identifying attributes (e.g., location) of sufficient users in the system so that it can generate valid queries identifying them. Eliminating the trusted proxy and generating dummy queries on the client is possible, but requires distributed mechanisms that allow each client to know the identifying attributes of other clients in the system, which in turn requires clients to trust each other.

2.3 Confidentiality

A promising alternative is for the client to eschew k-anonymity and instead use randomly generated dummy queries to provide a different guarantee, in which the service S knows exactly who issued the query, but does not know which query was actually issued. In this case, the client simply generates a set of random dummy queries that do not necessarily correspond to other users in the system, but simply constitute valid, arbitrarily chosen inputs to the service (e.g., "X restaurants near GPS coordinates Y", where X is a random cuisine type and Y is a random street address) . Such schemes have been proposed in the context of DNS privacy [9, 10]. While these dummy queries can be generated by the device inexpensively without coordinating with other clients or a trusted proxy, the problem still remains this scheme introduces overhead that increases linearly with the degree of confidentiality required; a dummy query set of k messages results in k extra requests and responses, massively increasing the bandwidth usage, power consumption, and end-to-end latency of the original query.

2.4 PIR (Private Information Retrieval)

To make random dummy queries efficient, one avenue is Private Information Retrieval [6]. In the simplest form of PIR, the service is replicated twice, and the client makes a query to each replica with a set of inputs. The sets of inputs in the two queries are identical, except that one contains the actual input and the other doesn't. Each of the replicas returns a single XOR of the requested responses, which the client then combines to retrieve the response to its actual input.

PIR is defined as follows: a server S holds a database with n bits, X = (X1 . . .Xn). A user u has a particular index i and wishes to retrieve the value of Xi, without disclosing to S the value of i. The PIR concept was introduced in [6] in an information theoretic setting,

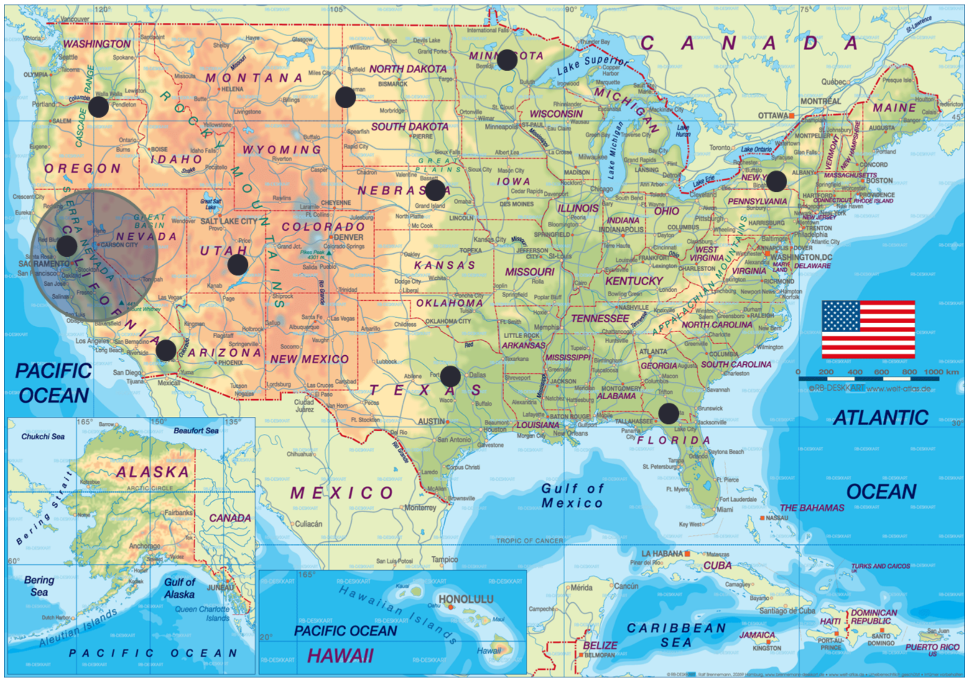


Figure 1: **Use of dummies and fuzzy location for location privacy.**

requiring that even if S had infinite computational power, it could not find i. It is proven in [6] that in any solution with a single server, you must receive the entire database. Nevertheless, in practice, it is sufficient to ensure that S cannot find i with polynomial-time computations; this problem is known as Computational PIR. The concept of PIR has been around since 1998, but it has been considered a more theoretical solution for database issues. In the recent years, the technology of cloud computing has shown tremendous potential and hence researchers have tried making PIR work with cloud to make it more practical. In 2008, Ghinita et al. studied a case similar to this project's – they studied the privacy of location-based information queried via mobile phones. They propose a framework for private location-dependent queries, which uses PIR protocols and eliminates the need for any trusted third party. They developed algorithms for approximate and exact private nearest neighbor search and utilized data mining techniques to optimize query execution. [11] Their experimental results suggest that PIR approaches incur reasonable overhead and are applicable in practice. They however, do not use a cloud as server but use a traditional server for the purpose of their study which might be able to reduce the high CPU cost they encounter.

Such a scheme has the advantage that it provides a strong guarantee – each replica of the service does not know which of the requested inputs is the actual one – in a very efficient manner, requiring the device to send two requests and receive two responses, regardless of the number of dummy inputs involved. However, PIR comes at a significant cost; it requires the service to be replicated, which may not be possible with real-world services such as Google Maps or Wikipedia. Additionally, PIR requires the service to be modified so it returns some compact function of the set of responses (an XOR in the case of the 2-replica scheme), which again may not be possible with third-party services.

3 The PriView Technique

PriView implements PIR without requiring the target service to be modified. It achieves this goal by inserting a layer of proxies in between the client device and the service. In effect, each proxy appears to the device as a replicated copy of the service, modified to return XORs instead of first-class responses.

3.1 The Basic scheme

- Step 1. When the client wants to issue a query A to a service, it first generates $K - 1$ other random dummy queries (K is 3 in the figure, and the dummy queries are B and C). It then sends a message requesting the set of queries $\{A, B, C\}$ to both proxies. *Each proxy can see the identity of the client but does not know which of the three queries is the actual one.*

- Step 2./3. Each proxy separately issues each of the queries A, B and C to the target service and receives the responses A', B', and C'. *The target service does not know the identity of the client, and cannot determine which of the three queries is the actual one.*

- Step 4. When a proxy gets back all the responses from the service, it combines a subset of them into an XOR. A bitmask determining this subset is provided by the client to the proxy in each request message. The subsets XORed by the two proxies are identical, except that one contains the actual query and the other doesn't. For example, in the figure, one proxy returns $\{A', B'\}$ while the other returns $\{B'\}$. *Whether a particular query's response is included in the XOR or not does not reveal any information to the proxy, since it could be a dummy query in either case: if it is included in the XOR, it could also be included in the other proxy's XOR, and if it's not, it might be missing in the other proxy's XOR.*

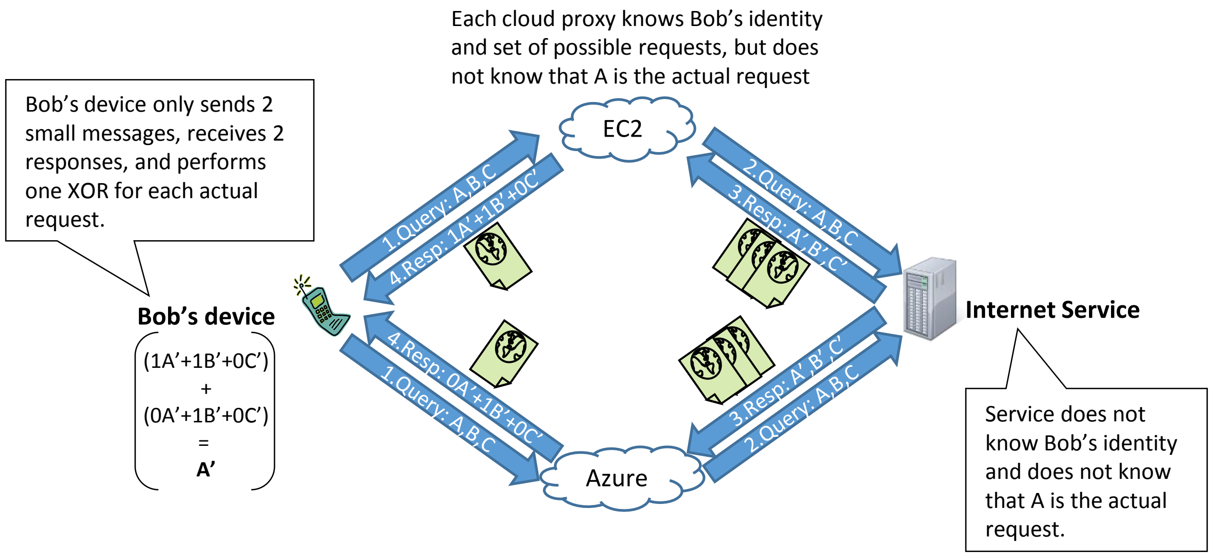

Each cloud proxy knows Bob's identity and set of possible requests, but does not know that A is the actual request
Bob's device only sends 2 small messages, receives 2 responses, and performs one XOR for each actual request.
EC2
1.Query: A,B,C
4.Resp: 1A'+1B'+0C'
2.Query: A,B,C
3.Resp: A',B',C'
Bob's device
(1A'+1B'+0C')
+
(0A'+1B'+0C')
=
A'
Internet Service
4.Resp: 0A'+1B'+0C'
1.Query: A,B,C
3.Resp: A',B',C'
2.Query: A,B,C
Azure
Service does not know Bob's identity and does not know that A is the actual request.

Figure 2: **The PriView technique: each proxy accepts a set of dummy queries from the client, issues them to the service, and returns an XOR of a subset of the responses to the client. The subsets are chosen so that the actual response required (A' in this example) can be reconstructed by the client.**

When the client receives both XORs, it combines them to reconstruct the response to the actual query. In figure 1., $(A' + B') + (B') = A'$.

This simple scheme provides the client data confidentiality with respect to the proxies. In other words, the proxy knows the identity of the client and a set of possible queries, but does not know exactly which query was issued by the client. It provides both confidentiality and anonymity with respect to the target service, which knows neither the identity of the client device or the exact query it issued.

In terms of efficiency, each first-class query issued by the client results in two outgoing messages, containing a list of K queries and a bitmask of K bits determining the subset to be returned, and two incoming messages, each of which is an XOR of multiple responses. Increasing the value of K provides a higher degree of confidentiality without increasing overhead significantly at the client; each additional dummy query results in a few extra bytes on the outgoing message. The size of the response XOR only increases if the newly added dummy query happens to generate the largest response out of all the queries.

3.2 Multiple proxies for lower latency

In the two-proxy scheme described above, the client has to always wait for the slowest of the two proxies to respond before reconstructing the answer. Using more proxies can mitigate this problem, with each proxy returning a linear combination of the responses it receives (rather than the XOR of a subset as before). The linear combinations generated by different proxies are identical, except that they use different coefficients for the actual query response. This allows the client to reconstruct the actual query response from the first two proxies that respond. For example, if a device wants to make an actual query A, it constructs a dummy set of queries $\{A, B, C, D\}$ and sends this to three proxies P_1 and P_2 and P_3. The proxies then return:

P_1 returns $A' + \alpha B' + \beta C' + \gamma D'$

P_2 returns $2A' + \alpha B' + \beta C' + \gamma D'$

P_3 returns $3A' + \alpha B' + \beta C' + \gamma D'$

The client can then extract the value of A' from these two linear combinations, since B', C' and D' cancel out by virtue of having equal coefficients. To implement this scheme, the client includes a set of coefficients in its request message to the proxies, instead of a bitmask as before. Note that the simpler two-proxy XOR-of-subset scheme is equivalent to the case where the linear combination exclusively uses binary coefficients; the bitmask can be thought of as a set of binary coefficients.

Unfortunately, it does not provide confidentiality with respect to the service. Since the service is not replicated, it can observe all incoming queries and infer the actual query. In Figure 2, the service sees the following sequence of requests arrive in a short time span: (A, B, C, B, C). From this sequence, it can infer that A is the actual query made by the client, even though it doesn't know the network address of the client. For some applications, knowing the actual query is often enough to compromise the identity of the user (as in the previously mentioned example where the user searches for restaurants near her house).

To support confidentiality for such applications, we modify the PriView technique to use linear combinations of responses instead of simple XORs. In this scheme, the dummy query sets sent to each proxy are perfectly identical; they all include the actual query, unlike the XOR-based scheme where one of the sets did not include the actual query. The proxies no longer reply with simple XORs of the query responses; instead, they return linear equations of the query responses, where the coefficients of each dummy response are identical across proxies but the coefficients of the actual response are different. The device can then solve the resulting system of linear equations to extract the actual response.

The resulting scheme provides confidentiality with respect to the proxies, which see a set of queries as in the earlier XOR-based scheme, as well as confidentiality with respect to the service, which now sees the stream of requests (A, B, C, A, B, C) from the two proxies and hence cannot determine that A is the actual query being made.

Linear combinations also enable us to use more than two proxies to reduce request latency. In the two-proxy scheme, the client has to always wait for the slowest of the two proxies to respond before reconstructing the answer. Using more proxies can mitigate this problem, with each proxy returning a linear combination with a different coefficient for the actual query response, so that the client only has to wait for the first two proxies to respond. For example, in addition to proxies P_1 and P_2 above, the client might also use proxy P_3, which returns:

P_3 returns $3A' + \alpha B' + \beta C' + \gamma D'$

The latency benefit of using more than two proxies comes at a cost. It requires the device to send out more network packets for each query, one per proxy. In addition, the device receives multiple responses back, one from each proxy; it can attempt to short-circuit these responses by notifying the remaining proxies once it has heard back from the first two, this requires more outgoing packets that may not always reach the proxies in time. Each wasted response increases the bandwidth and battery overhead for the query.

This approach results in improved latency, since the device now has to only wait for the fastest two responders out of a potentially large number of proxies. However, it has a number of drawbacks. First, it requires more complex and battery-draining logic on the device to solve linear equations, rather than the simple XOR operations of the two-proxy solution. Second, it requires the device to send out more network packets for each query, one per proxy. Third, the device receives multiple responses back, one from each proxy; it can attempt to short circuit these responses by notifying the remaining proxies once it has heard back from the first two, but each extra proxy that responds increases the bandwidth and battery overhead for the query.

The downside of using linear combinations instead of simple XORs is increased complexity and possibly higher power consumption on the client device. In this paper, we implement and evaluate the XOR-based scheme; in the future, we intend to quantify the performance and power implications of using linear combinations.

3.3 Generating dummy queries

The PriView technique requires an effective, inexpensive way to generate random dummy queries. Some services provide hooks for generating random queries; for example, Wikipedia provides a special URL (`http://en.wikipedia.org/wiki/Special:Random`) that returns the article corresponding to a random query. For other applications, random queries can be generated algorithmically; for example, a Google Maps query is simply a latitude/longitude pair.

However, a truly random dummy query can fail to provide sufficient privacy. If a proxy is presented with a dummy set of multiple latitude/longitude pairs, of which one is in a major urban center while the others are in less populated areas, it can deduce that the former is the actual query being made. Accordingly, care has to be taken that dummy queries are indistinguishable from actual queries; in the case of geographical locations, this can be achieved by biasing the sampling process, such that the probability that a location is chosen is proportional to its population density.

3.4 Limitations

PriView does have limitations. It requires the Internet service to respond deterministically to a given query; if the two cloud proxies can get different responses for the same dummy queries, the device can no longer retrieve the original query's response by combining the two returned XORs. In addition, PriView is not relevant for services that require a user login and store user-specific state, such as email or social networks; in this case, the service already knows the identity of the user. Finally, as described above, it assumes that an efficient means exists for generating random dummy queries to the service.

More than two proxies: PriView adds latency to queries by interposing proxies between the device and the Internet service. Not only does this add an extra hop to the query path, but the device now has to wait for both proxies to respond, allowing query latency to be determined by the slower of the two proxies.

To mitigate the problem of increased latency, PriView allows the use of more than two proxies; the device sends a set of dummy queries to each proxies, and then waits for the first two that reply. This multi-proxy scheme is different from the two-proxy case in two ways. First, the dummy query sets sent to each proxy are perfectly identical; they all include the actual query, unlike the two-proxy case where one of the sets did not include the actual query. Second, the proxies no longer reply with simple XORs of the query responses; instead, they return linear equations of the query responses, where the coefficients of each dummy response are identical across proxies but the coefficients of the actual response are different. The device can then solve the resulting system of linear equations to extract the actual response.

For example, if a device wants to make an actual query A, it constructs a dummy set of queries (A, B, C, D) and sends this to three proxies (P_1, P_2, P_3). If the response for A is A', the proxies then return:

P_1 returns $A' + \alpha B' + \beta C' + \gamma D'$

P_2 returns $2A' + \alpha B' + \beta C' + \gamma D'$

Any two of these three equations can be used to reconstruct the value of A', since B', C' and D' cancel out by virtue of having equal coefficients.

3.5 Network attacks

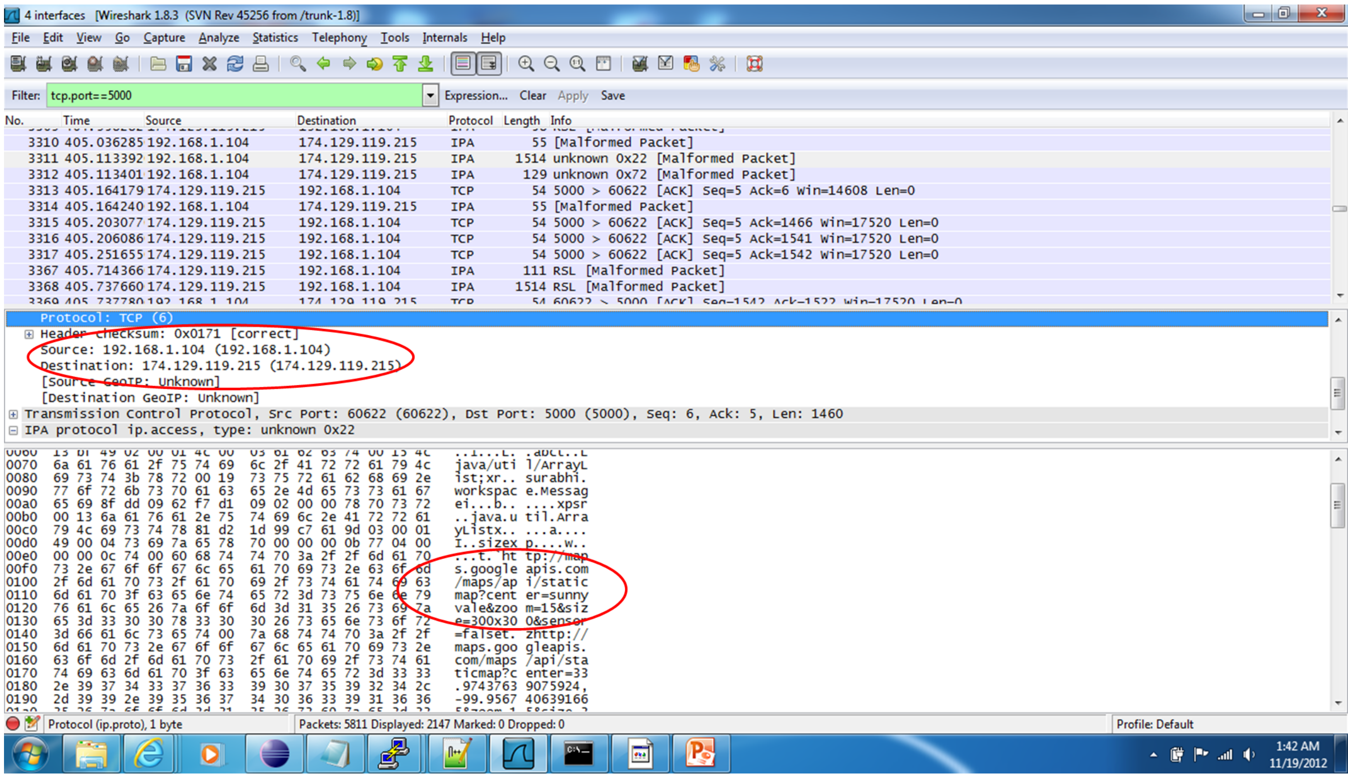
We implemented two security attacks on PriView. Each of these attacks can be perpetrated at the two levels of communication: between the application or user and the cloud and between the cloud and the query server.

3.5.1 Attack on client-cloud connection

We implemented a man-in-the-middle attack in the communication between the user and the cloud. This way the attacker will be able to eavesdrop on the connection thus making the private querying process no more private.

Man-in-the-Middle: The term ”Man-in-the-middle attack” (MITM attack) refers to the type of attack where the attacker intrudes into the communication between the endpoints on a network to inject false information and intercept the data transferred between them. In cryptography, the man-in-the-middle attack or bucket-brigade attack (often abbreviated MITM), sometimes Janus attack, is a form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection when in fact the entire conversation is controlled by the attacker. One of the objectives for MITM attacks is to gain access to the client's messages and modify them before finally transmitting them to the server end. Other objectives of MITM can be to mislead the communicators at the client or server end, to intercept pertinent information (e.g., identity, address, password, or any other confidential information for malicious purposes) and also, at times, manipulate transactions.

In this case, we are using MITM to intercept and eavesdrop on the location information that the user sends to the cloud. The way these queries are formed is following - one of the queries is entered by the user through the text box in the Android application. The user (client) code then generates n number of random coordinates within a defined coordinate space. The actual query, a string, is different from the randomly generated numbers as coordinates. With a freely available sniffing software like Wireshark, any malicious attacker using the same network as the client phone will be able to see all the packets coming and going to and from all IP addresses on that network. This scenario can easily be envisioned in a WiFi hotspot, like a cafe. We join the same free network on our phone as several other people in the cafe. We use our phone to scout the map of the area on Google Maps, and we use the secure application. We type the query in the textbox and hit send. While we are using TCP socket connections in the code, the packets are still unencrypted. A Wireshark


4 interfaces [Wireshark 1.8.3 (SVN Rev 45256 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp.port==5000
Expression... Clear Apply Save
Profile: Default
1:42 AM
11/19/2012

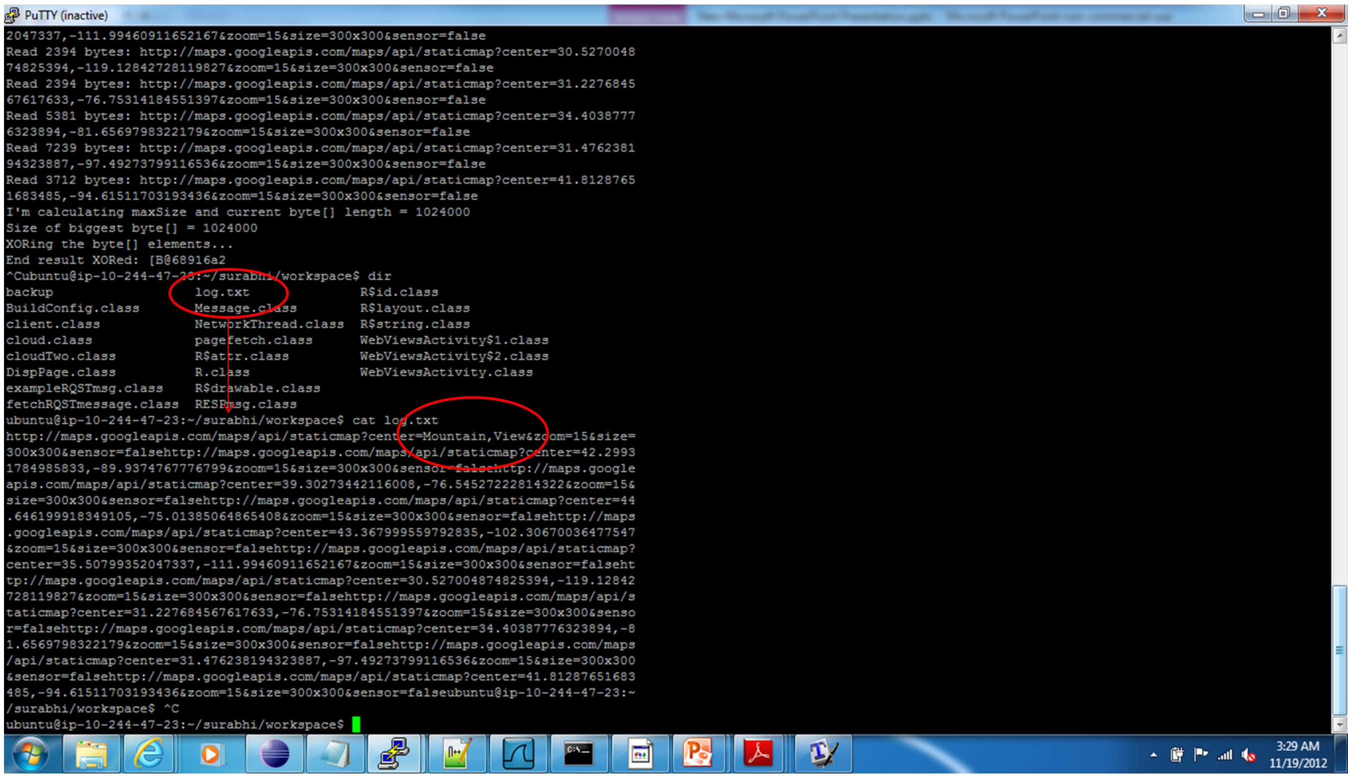
```
No.   Time        Source           Destination      Protocol Length Info
 3310 405.036285 192.168.1.104    174.129.119.215  IPA        55 [Malformed Packet]
 3311 405.113392 192.168.1.104    174.129.119.215  IPA      1514 unknown 0x22 [Malformed Packet]
 3312 405.113401 192.168.1.104    174.129.119.215  IPA       129 unknown 0x72 [Malformed Packet]
 3313 405.164179 174.129.119.215  192.168.1.104    TCP        54 5000 > 60622 [ACK] Seq=5 Ack=6 Win=14608 Len=0
 3314 405.164240 192.168.1.104    174.129.119.215  IPA        55 [Malformed Packet]
 3315 405.203077 174.129.119.215  192.168.1.104    TCP        54 5000 > 60622 [ACK] Seq=5 Ack=1466 Win=17520 Len=0
 3316 405.206086 174.129.119.215  192.168.1.104    TCP        54 5000 > 60622 [ACK] Seq=5 Ack=1541 Win=17520 Len=0
 3317 405.251655 174.129.119.215  192.168.1.104    TCP        54 5000 > 60622 [ACK] Seq=5 Ack=1542 Win=17520 Len=0
 3367 405.714366 174.129.119.215  192.168.1.104    IPA       111 RSL  [Malformed Packet]
 3368 405.737660 174.129.119.215  192.168.1.104    IPA      1514 RSL  [Malformed Packet]

    Protocol: TCP (6)
  ⊞ Header checksum: 0x0171 [correct]
    Source: 192.168.1.104 (192.168.1.104)
    Destination: 174.129.119.215 (174.129.119.215)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊞ Transmission Control Protocol, Src Port: 60622 (60622), Dst Port: 5000 (5000), Seq: 6, Ack: 5, Len: 1460
⊟ IPA protocol ip.access, type: unknown 0x22

0070  6a 61 76 61 2f 75 74 69  6c 2f 41 72 72 61 79 4c   java/uti l/ArrayL
0080  69 73 74 3b 78 72 00 19  73 75 72 61 62 68 69 2e   ist;xr.. surabhi.
0090  77 6f 72 6b 73 70 61 63  65 2e 4d 65 73 73 61 67   workspac e.Messag
00a0  65 69 8f dd 09 62 f7 d1  09 02 00 00 78 70 73 72   ei...b.. ....xpsr
00b0  00 13 6a 61 76 61 2e 75  74 69 6c 2e 41 72 72 61   ..java.u til.Arra
00c0  79 4c 69 73 74 78 81 d2  1d 99 c7 61 9d 03 00 01   yListx.. ...a....
00d0  49 00 04 73 69 7a 65 78  70 00 00 00 0b 77 04 00   I..sizex p....w..
00e0  00 00 0c 74 00 60 68 74  74 70 3a 2f 2f 6d 61 70   ...t.`ht tp://map
00f0  73 2e 67 6f 6f 67 6c 65  61 70 69 73 2e 63 6f 6d   s.google apis.com
0100  2f 6d 61 70 73 2f 61 70  69 2f 73 74 61 74 69 63   /maps/ap i/static
0110  6d 61 70 3f 63 65 6e 74  65 72 3d 73 75 6e 6e 79   map?cent er=sunny
0120  76 61 6c 65 26 7a 6f 6f  6d 3d 31 35 26 73 69 7a   vale&zoo m=15&siz
0130  65 3d 33 30 30 78 33 30  30 26 73 65 6e 73 6f 72   e=300x30 0&sensor
0140  3d 66 61 6c 73 65 74 00  7a 68 74 74 70 3a 2f 2f   =falset. zhttp://
0150  6d 61 70 73 2e 67 6f 6f  67 6c 65 61 70 69 73 2e   maps.goo gleapis.
0160  63 6f 6d 2f 6d 61 70 73  2f 61 70 69 2f 73 74 61   com/maps /api/sta
0170  74 69 63 6d 61 70 3f 63  65 6e 74 65 72 3d 33 33   ticmap?c enter=33
0180  2e 39 37 34 33 37 36 33  39 30 37 35 39 32 34 2c   .9743763 9075924,
0190  2d 39 39 2e 39 35 36 37  34 30 36 33 39 31 36 36   -99.9567 40639166

Protocol (ip.proto), 1 byte    Packets: 5811 Displayed: 2147 Marked: 0 Dropped: 0
```

Figure 3: **MITM using Wireshark.**



```
2047337,-111.99460911652167&zoom=15&size=300x300&sensor=false
Read 2394 bytes: http://maps.googleapis.com/maps/api/staticmap?center=30.5270048
74825394,-119.12842728119827&zoom=15&size=300x300&sensor=false
Read 2394 bytes: http://maps.googleapis.com/maps/api/staticmap?center=31.2276845
67617633,-76.75314184551397&zoom=15&size=300x300&sensor=false
Read 5381 bytes: http://maps.googleapis.com/maps/api/staticmap?center=34.4038777
6323894,-81.6569798322179&zoom=15&size=300x300&sensor=false
Read 7239 bytes: http://maps.googleapis.com/maps/api/staticmap?center=31.4762381
94323887,-97.49273799116536&zoom=15&size=300x300&sensor=false
Read 3712 bytes: http://maps.googleapis.com/maps/api/staticmap?center=41.8128765
1683485,-94.61511703193436&zoom=15&size=300x300&sensor=false
I'm calculating maxSize and current byte[] length = 1024000
Size of biggest byte[] = 1024000
XORing the byte[] elements...
End result XORed: [B@68916a2
^Cubuntu@ip-10-244-47-23:~/surabhi/workspace$ dir
backup                 log.txt               R$id.class
BuildConfig.class      Message.class         R$layout.class
client.class           NetworkThread.class   R$string.class
cloud.class            pagefetch.class       WebViewsActivity$1.class
cloudTwo.class         R$attr.class          WebViewsActivity$2.class
DispPage.class         R.class               WebViewsActivity.class
exampleRQSTmsg.class   R$drawable.class
fetchRQSTmessage.class RESPmsg.class
ubuntu@ip-10-244-47-23:~/surabhi/workspace$ cat log.txt
http://maps.googleapis.com/maps/api/staticmap?center=Mountain,View&zoom=15&size=
300x300&sensor=falsehttp://maps.googleapis.com/maps/api/staticmap?center=42.2993
1784985833,-89.93747677776799&zoom=15&size=300x300&sensor=falsehttp://maps.google
apis.com/maps/api/staticmap?center=39.30273442116008,-76.54527222814322&zoom=15&
size=300x300&sensor=falsehttp://maps.googleapis.com/maps/api/staticmap?center=44
.646199918349105,-75.01385064865408&zoom=15&size=300x300&sensor=falsehttp://maps
.googleapis.com/maps/api/staticmap?center=43.367999559792835,-102.30670036477547
&zoom=15&size=300x300&sensor=falsehttp://maps.googleapis.com/maps/api/staticmap?
center=35.50799352047337,-111.99460911652167&zoom=15&size=300x300&sensor=falseht
tp://maps.googleapis.com/maps/api/staticmap?center=30.527004874825394,-119.12842
728119827&zoom=15&size=300x300&sensor=falsehttp://maps.googleapis.com/maps/api/s
taticmap?center=31.227684567617633,-76.75314184551397&zoom=15&size=300x300&senso
r=falsehttp://maps.googleapis.com/maps/api/staticmap?center=34.40387776323894,-8
1.6569798322179&zoom=15&size=300x300&sensor=falsehttp://maps.googleapis.com/maps
/api/staticmap?center=31.47623819432387,-97.49273799116536&zoom=15&size=300x300
&sensor=falsehttp://maps.googleapis.com/maps/api/staticmap?center=41.81287651683
485,-94.61511703193436&zoom=15&size=300x300&sensor=falseubuntu@ip-10-244-47-23:~
/surabhi/workspace$ ^C
ubuntu@ip-10-244-47-23:~/surabhi/workspace$
```

Figure 4: **Log file generated by the tampered code**

user is monitoring the packets on the free network in the cafe and they can see which IP addresses are sending what information. In our case, the location string is easily visible, with the IP addresses of source and destination machines. Thus, the privacy of our query is compromised.

Suggested solution: There are two things that can be done to avoid the attack described above. (a.) Provide end-to-end authentication, and (b.) Encrypt the query going from the user to the cloud.

3.5.2 Attack on cloud-query server connection

One of the assumptions we made while creating this system was that the cloud instances used for querying and XORing the results do not talk to each other. Let us assume the cloud provider we are using has a vested interest in checking all the requests sent to and from the instances - for example, Amazon wants to personalize your holiday shopping experience and knows that you have an AWS account. It wants to monitor the requests made through this cloud to different query servers for any kind of clue. Amazon has access to your cloud code and all it needs to do is to insert a small piece of code in it that will generate a log file for all the requests made in all the instances you use. When it checks the two log files generated by the two instances we use for this project, it is very easy to see what the user is interested in it. Though highly unethical, if a cloud provider decides to use the information you store on their servers, they can.

As an experiment, we put in a small piece of code that created a log file in each instance. Figure 4 shows what the log file stored at the end of the run:

Suggested solution: The way to ensure that this does not happen is postulated in the solution originally proposed - use two different cloud providers so they see the set of queries each and can't know accurately which one is the query user is actually interested in. Another assumption made for this project was that the service was deterministic. We can try changing the content of the queries (for example, a Wikipedia page that is one of the queries can be edited) between the time taken by the two clouds to query the same page.

3.6 SSL Encryption

SSL(Secure Socket Layer) is a protocol developed by Netscape for transmitting private documents via the Internet. SSL uses a cryptographic system that uses two keys to encrypt data a public key known to everyone and a private or secret key known only to the recipient of the message. Both Netscape Navigator and Internet Explorer support SSL, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers. By convention, URLs that require an SSL connection start with https: instead of http.

SSL aims to provide applications with a secure socket like toolkit. SSL addresses three important security issues: 1.It provides authentication, which helps ensure the legitimacy of the entities involved in a dialog. 2.It provides privacy. SSL helps warrant that a third party cannot decipher the dialog between two entities. 3.It maintains integrity. The use of a MAC (message authentication code), which is similar to a checksum, helps guarantee that a dialog between two entities is not modified by a third party.

SSL relies heavily on both public-key and secret-key cryptography. It uses secret-key cryptography to bulk-encrypt the data exchanged between two applications. SSL provides the ideal solution because secret-key algorithms are both secure and fast. Public-key cryptography, which is slower than secret-key cryptography, is a better choice for authentication and key exchange.

3.6.1 SSLSocketFactory Class

SSLSocketFactory can be used to validate the identity of the HTTPS server against a list of trusted certificates and to authenticate to the HTTPS server using a private key. SSLSocketFactory will enable server authentication when supplied with a trust-store file containing one or several trusted certificates. The client secure socket will reject the connection during the SSL session handshake if the target HTTPS server attempts to authenticate itself with a non-trusted certificate. In special cases the standard trust verification process can be bypassed by using a custom TrustStrategy. This interface is primarily intended for allowing self-signed certificates to be accepted as trusted without having to add them to the trust-store file. The following sequence of actions is used to generate a key-store file:

(i) Use JDK keytool utility to generate a new key:

```
keytool-genkey-v-alias"myclientkey"-validity365-keystoremy.keystore
```

For simplicity use the same password for the key as that of the key-store.

(ii) Issue a certificate signing request (CSR):

```
keytool-certreq-alias"myclientkey"-filemycertreq.csr-keystoremy.
keystore
```

Send the certificate request to the trusted Certificate Authority for signature. One may choose to act as her own CA and sign the certificate request using a PKI tool, such as OpenSSL.

(iii) Import the trusted CA root certificate:

```
keytool-import-alias"mytrustedca"-filecaroot.crt-keystoremy.
keystore
```

(iv) Import the PKCS7 file containg the complete certificate chain:

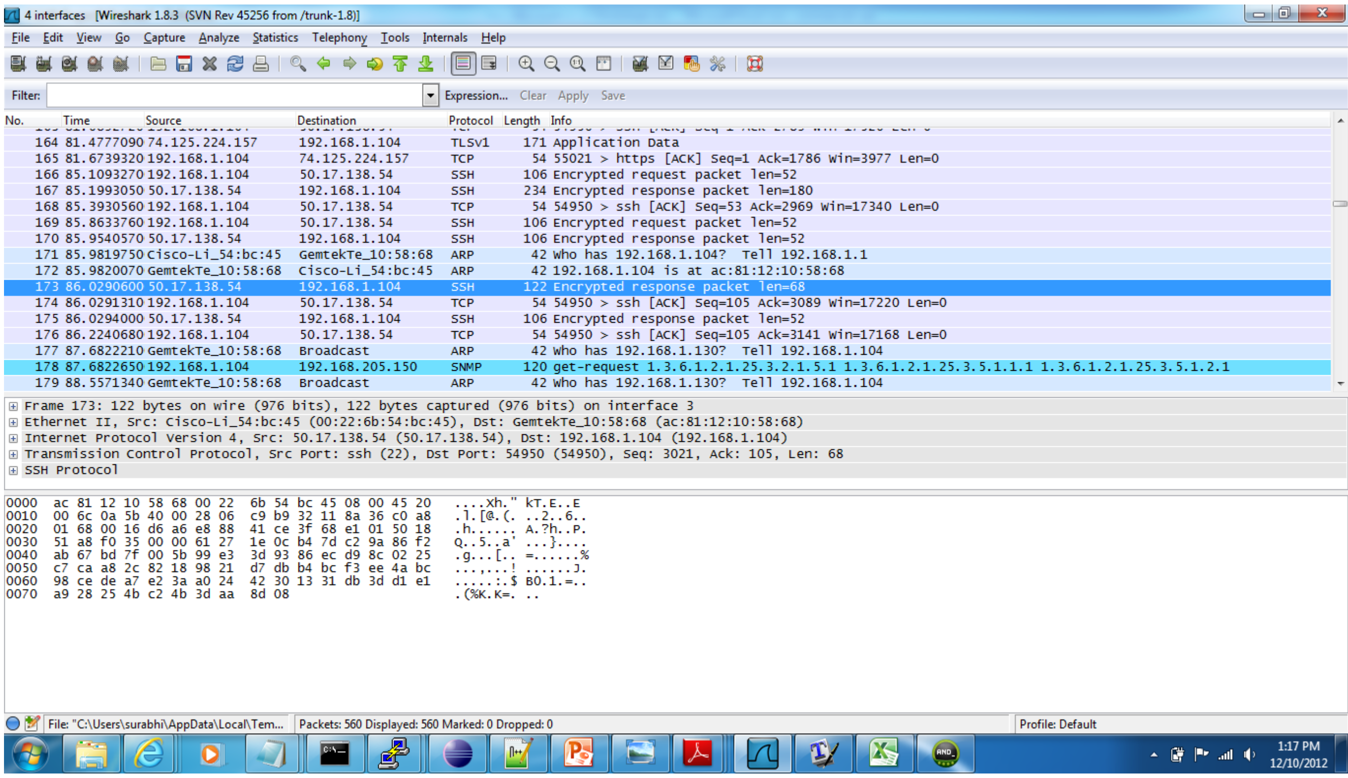
```
keytool-import-alias"myclientkey"-filemycert.p7-keystoremy.keystore
```

(v) Verify the content the resultant keystore file:

```
keytool-list-v-keystoremy.keystore
```

3.6.2 Implementation of SSL Sockets

To implement SSL sockets, we generated the key using the keytool and embedded it in the client code and copied it to each cloud instance. Also, a boolean was placed in the code so that a flip can be made between the normal Socket connection and the SSL connection. This was mainly to generate and compare unencrypted and encrypted data when needed. When this boolean was true, the connection was made by the SSL sockets. The client aunthenticates itself with the cloud using the stored keys. Then, encrypted requests are sent to the cloud and responses coming back from the cloud are encrypted as well. When the WireShark tool is used, as in stage 1, to sniff packets, it can be seen in figure 5 that all the transactions are encrypted.


4 interfaces [Wireshark 1.8.3 (SVN Rev 45256 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.     Time       Source             Destination        Protocol Length Info
    164 81.4777090 74.125.224.157     192.168.1.104      TLSv1     171 Application Data
    165 81.6739320 192.168.1.104      74.125.224.157     TCP        54 55021 > https [ACK] Seq=1 Ack=1786 Win=3977 Len=0
    166 85.1093270 192.168.1.104      50.17.138.54       SSH       106 Encrypted request packet len=52
    167 85.1993050 50.17.138.54       192.168.1.104      SSH       234 Encrypted response packet len=180
    168 85.3930560 192.168.1.104      50.17.138.54       TCP        54 54950 > ssh [ACK] Seq=53 Ack=2969 Win=17340 Len=0
    169 85.8633760 192.168.1.104      50.17.138.54       SSH       106 Encrypted request packet len=52
    170 85.9540570 50.17.138.54       192.168.1.104      SSH       106 Encrypted response packet len=52
    171 85.9819750 Cisco-Li_54:bc:45  GemtekTe_10:58:68  ARP        42 Who has 192.168.1.104?  Tell 192.168.1.1
    172 85.9820070 GemtekTe_10:58:68  Cisco-Li_54:bc:45  ARP        42 192.168.1.104 is at ac:81:12:10:58:68
    173 86.0290600 50.17.138.54       192.168.1.104      SSH       122 Encrypted response packet len=68
    174 86.0291310 192.168.1.104      50.17.138.54       TCP        54 54950 > ssh [ACK] Seq=105 Ack=3089 Win=17220 Len=0
    175 86.0294000 50.17.138.54       192.168.1.104      SSH       106 Encrypted response packet len=52
    176 86.2240680 192.168.1.104      50.17.138.54       TCP        54 54950 > ssh [ACK] Seq=105 Ack=3141 Win=17168 Len=0
    177 87.6822210 GemtekTe_10:58:68  Broadcast          ARP        42 Who has 192.168.1.130?  Tell 192.168.1.104
    178 87.6822650 192.168.1.104      192.168.205.150    SNMP      120 get-request 1.3.6.1.2.1.25.3.2.1.5.1 1.3.6.1.2.1.25.3.5.1.1 1.3.6.1.2.1.25.3.5.1.2.1
    179 88.5571340 GemtekTe_10:58:68  Broadcast          ARP        42 Who has 192.168.1.130?  Tell 192.168.1.104

⊞ Frame 173: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface 3
⊞ Ethernet II, Src: Cisco-Li_54:bc:45 (00:22:6b:54:bc:45), Dst: GemtekTe_10:58:68 (ac:81:12:10:58:68)
⊞ Internet Protocol Version 4, Src: 50.17.138.54 (50.17.138.54), Dst: 192.168.1.104 (192.168.1.104)
⊞ Transmission Control Protocol, Src Port: ssh (22), Dst Port: 54950 (54950), Seq: 3021, Ack: 105, Len: 68
⊞ SSH Protocol

0000  ac 81 12 10 58 68 00 22  6b 54 bc 45 08 00 45 20   ....Xh." kT.E..E
0010  00 6c 0a 5b 40 00 28 06  c9 b9 32 11 8a 36 c0 a8   .l.[@.(. ..2..6..
0020  01 68 00 16 d6 a6 e8 88  41 ce 3f 68 e1 01 50 18   .h...... A.?h..P.
0030  51 a8 f0 35 00 00 61 27  1e 0c b4 7d c2 9a 86 f2   Q..5..a' ...}....
0040  ab 67 bd 7f 00 5b 99 e3  3d 93 86 ec d9 8c 02 25   .g...[.. =......%
0050  c7 ca a8 2c 82 18 98 21  d7 db b4 bc f3 ee 4a bc   ...,...! ......J.
0060  98 ce de a7 e2 3a a0 24  42 30 13 31 db 3d d1 e1   .....:.$ B0.1.=..
0070  a9 28 25 4b c2 4b 3d aa  8d 08                     .(%K.K=. ..

File: "C:\Users\surabhi\AppData\Local\Tem...   Packets: 560 Displayed: 560 Marked: 0 Dropped: 0   Profile: Default
1:17 PM
12/10/2012
```

Figure 5: **Encrypted requests and responses.**

4 Evaluation

4.1 Experimental setup

We implemented an Android-based PriView client for the Google Static Maps [7] service, with proxies running on Amazon EC2. We evaluated this client on a MK802 Android Mini-PC running Android 4.0.3, as well as a Samsung S5570 Galaxy Mini running Android 2.3.3. Our comparison point was a client that issued multiple dummy requests for each query directly from the device. Both PriView and this strawman client offer the same confidentiality guarantee: if we use K dummy requests for each actual query, the service cannot tell which of $K + 1$ requests the device actually made. We call K the degree of confidentiality. The strawman client running with degree of confidentiality $K = 0$ corresponds to an unmodified, conventional client that does not provide any confidentiality. We do not plot $K = 0$ for PriView, since our mechanism requires at least one dummy query to be issued.

4.2 Bandwidth usage

Figure 6 shows the average bandwidth used by each query as we increase the degree of confidentiality by issuing more dummy requests per query. As PriView issues more dummy requests, the size of its outgoing messages go up linearly while the size of the returning messages increases sub-linearly; each returning message is an XOR of multiple responses, hence its size is equal to that of the largest response. Since each outgoing message is simply a list of dummy inputs to the service (e.g., GPS coordinates or Wikipedia URLs), it is much smaller than the returning message, which can be roughly 25 KB in the case of Google Static Maps. As a result, total bandwidth used by PriView increases sub-linearly as the number of dummy requests is increased and the degree of confidentiality rises.

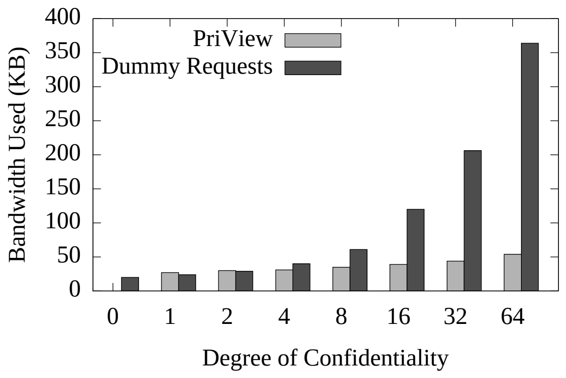


Figure 6: **As the degree of confidentiality (i.e., the total number of requests seen by the service per query) is increased, bandwidth increases sub-linearly for PriView as request packets become larger, and linearly for conventional dummy queries.**

In contrast, we expect our comparison strawman to show a linear increase in bandwidth usage, since it sends and receives $K + 1$ distinct requests and responses for each query, in order to get a degree of confidentiality equal to K. Interestingly, we instead see sub-linear scaling of bandwidth for small numbers of dummy queries; for example, going from $K = 1$ to $K = 2$ does not double bandwidth used. This is an artefact of naive dummy query generation; since we pick random GPS coordinates for each dummy query, they are likely to fall on the ocean and return a highly compressible image of 2 to 3KB with no features. As a result, adding dummy queries does not necessarily double bandwidth for the strawman. However, if we used more intelligent dummy query generation based on population density, the returned maps would be feature-rich and non-compressible, in which case we would observe linear scaling of bandwidth. In summary, this graph makes two important points: PriView is bandwidth-efficient compared to conventional dummy requests, and it does not add much bandwidth overhead over an unmodified, non-confidential client (as seen by the $K = 0$ data point).

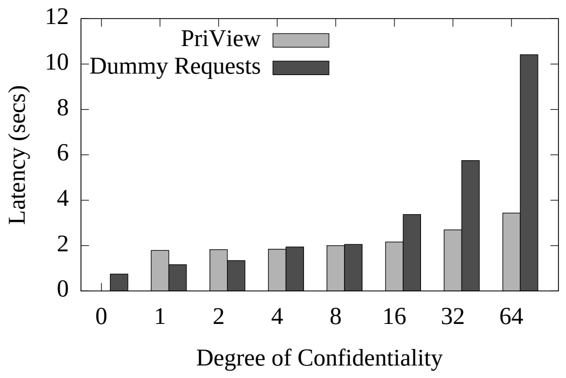


Figure 7: **As the degree of confidentiality is increased, latency increases sub-linearly for PriView.**

4.3 End-to-end delay

Figure 7 shows a similar trend for end-to-end latency. With the degree of confidentiality at 1, the number of messages at the device is the same for both approaches, but PriView has higher latency since it routes messages through its cloud proxies while the strawman directly accesses the service. PriView begins to outperform the strawman once the degree of confidentiality is reasonably high, since its large bandwidth advantage over the strawman outweighs the delay imposed by its proxy layer.

4.4 Power consumption

Finally, Figure 8 plots the power consumed per query for PriView and the strawman. Since the MK802 device we used does not have an internal battery and relies on external AC power, we were able to measure instantaneous power draw with a Kill A Watt [12] monitor. As shown in the graph, PriView's power consumption stays nearly constant even as the degree of confidentiality is increased, while the strawman client's power consumption increases rapidly.

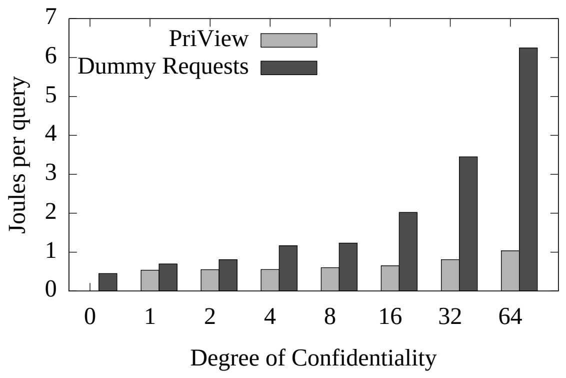


Figure 8: **Power consumption increases sub-linearly for PriView as degree of confidentiality is increased.**

5 Conclusion

Enabling privacy-preserving access to existing Internet services from mobile devices is a challenging problem: devices are resource-constrained, while services cannot easily be modified or replicated. PriView uses a novel approach to Private Information Retrieval that utilizes cloud-based proxies to provide confidentiality, without imposing high overhead on the device or requiring the target service to be modified or replicated. Our evaluation showed that PriView works with existing services and provides tunable confidentiality that imposes a sub-linear cost in terms of latency, bandwidth usage and power consumption.

In this project, we focused on fine-tuning the foundations of PriView and developing a robust and functional solution to the problem of query privacy. To further reinforce the privacy aspect and improve the scalability of the system, we plan to calibrate Priview to make further improvements, as part of our future work. A few of them are: addition of multiple proxies using different cloud providers, improving the quality of the dummy queries and adding multi-service capability to extend the system into a general-purpose, secure querying platform. Ultimately, PriViews goal is to provide a general-purpose, privacy-preserving querying platform for real-world services.

References

[1] “Google Glass.” [Online]. Available: http://www.google.com/glass/start/

[2] H. Kido, Y. Yanagisawa, and T. Satoh, “Protection of location privacy using dummies for location-based services,” in *Data Engineering Workshops, 2005. 21st International Conference on*. IEEE, 2005, pp. 1248–1248.

[3] S. Amini, J. Lindqvist, J. I. Hong, M. Mou, R. Raheja, J. Lin, N. Sadeh, and E. Tochb, “Caché: caching location-enhanced content to improve user privacy,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 14, no. 3, pp. 19–21, 2010.

[4] “Amazon EC2,” http://aws.amazon.com/ec2/.

[5] “”microsoft windows azure”,” http://www.windowsazure.com/en-us/.

[6] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, “Private information retrieval,” *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 965–981, 1998.

[7] “Google static maps,” https://developers.google.com/maps/documentation/staticmaps/.

[8] L. Sweeney, “k-anonymity: A model for protecting privacy,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[9] F. Zhao, Y. Hori, and K. Sakurai, “Analysis of privacy disclosure in dns query,” in *Multimedia and Ubiquitous Engineering, 2007. MUE’07. International Conference on*. IEEE, 2007, pp. 952–957.

[10] H. Federrath, K.-P. Fuchs, D. Herrmann, and C. Piosecny, “Privacy-preserving dns: analysis of broadcast, range queries and mix-based protection methods,” in *Computer Security–ESORICS 2011*. Springer, 2011, pp. 665–683.

[11] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, “Private queries in location based services: anonymizers are not necessary,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 121–132.

[12] “P4400 Kill A Watt,” http://www.p3international.com/products/P4400.html.