Hiding behind the Clouds: Efficient, Privacy-Preserving Queries via Cloud Proxies

Surabhi Gaur[1], Melody Moh[1], Mahesh Balakrishnan[2]
[1]Department of Computer Science, San Jose State University
[2]Microsoft Research Silicon Valley

Abstract—**This paper 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.**

I. 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.

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.

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.

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 paper 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.

II. 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:

A. 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.

B. 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.

C. 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.

D. 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. 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.

III. 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.

A. The Basic scheme

- Step 1. When the client wants to issue a query A to a service, it first generates K other random dummy queries (K is 2 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 tell the proxy if it's the actual query or a dummy query: 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.*

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.

A subtle point is that the size of the XORed subset returned by a proxy has to be exactly half of $K + 1$ for this scheme to provide optimal confidentiality (i.e., for the proxy to view each query as having a $\frac{1}{K+1}$ chance of being the actual query). To understand why, consider the information available to each proxy: a set of $K + 1$ queries that it must issue to the target service, one of which is the actual query, and a T-sized subset whose responses it must XOR and return to the client. The proxy knows that the actual query is in the T-sized subset with probability $\frac{1}{2}$ (since there are two proxies, one of which includes the actual query's response in its XOR). Accordingly, the probability that a particular element in the XORed subset is the actual query is $\frac{1}{2T}$; if T is not equal to $\frac{K+1}{2}$, this probability is greater than or less than $\frac{1}{K+1}$, allowing the proxy to know that some queries are more likely than others to be the actual query. In the limit, if T is equal to 1 or K, the proxy can determine that a particular query has a 50% chance of being the actual query. Conversely, if T is equal to $\frac{K+1}{2}$, the probability of a particular element being the actual query is $\frac{1}{K+1}$, which means that all $K + 1$ queries are equally likely to be the actual query from the viewpoint of the proxy.

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.

B. 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:

$$\begin{aligned} &P_1 \text{ returns } \; A' + \alpha B' + \beta C' + \gamma D' \\ &P_2 \text{ returns } 2A' + \alpha B' + \beta C' + \gamma D' \\ &P_3 \text{ returns } 3A' + \alpha B' + \beta C' + \gamma D' \end{aligned}$$

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.

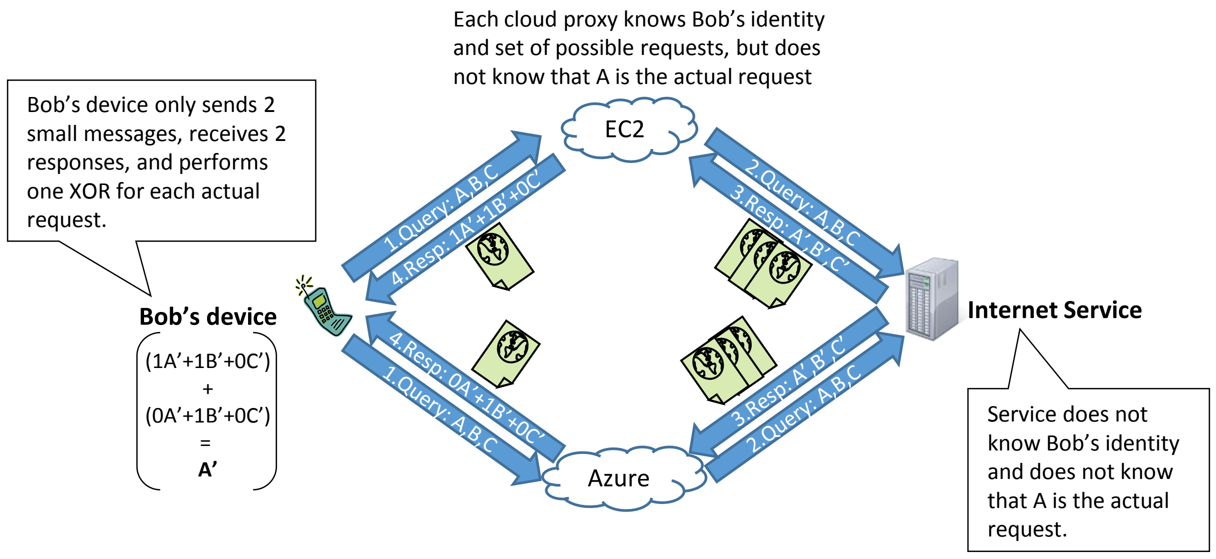
Each cloud proxy knows Bob's identity and set of possible requests, but does not know that A is the actual request
Bob's device only sends 2 small messages, receives 2 responses, and performs one XOR for each actual request.
EC2
1.Query: A,B,C
4.Resp: 1A'+1B'+0C'
2.Query: A,B,C
3.Resp: A',B',C'
Bob's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

Fig. 1. **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.**

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.

C. 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.

D. 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.

IV. Evaluation

A. 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.

B. Bandwidth usage

Figure 2 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

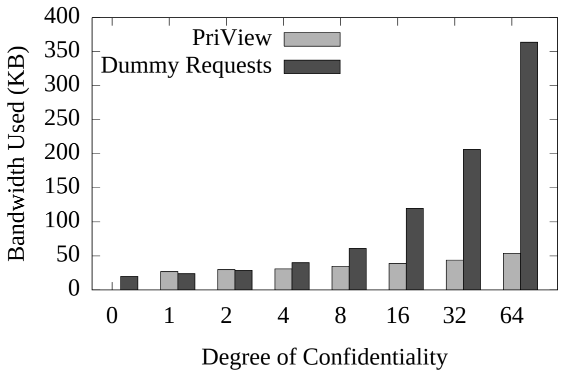

Fig. 2. **As the degree of confidentiality (i.e., the total number of requests seen by the service per query) is increased, bandwidth increases sub-linearly for PriView as request packets become larger, and linearly for conventional dummy queries.**

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.

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).

C. End-to-end delay

Figure 3 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.

D. Power consumption

Finally, Figure 4 plots the power consumed per query for PriView and the strawman. Since the MK802 device we used

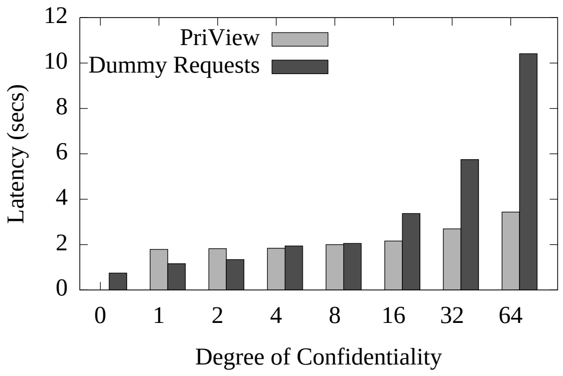

Fig. 3. **As the degree of confidentiality is increased, latency increases sub-linearly for PriView.**

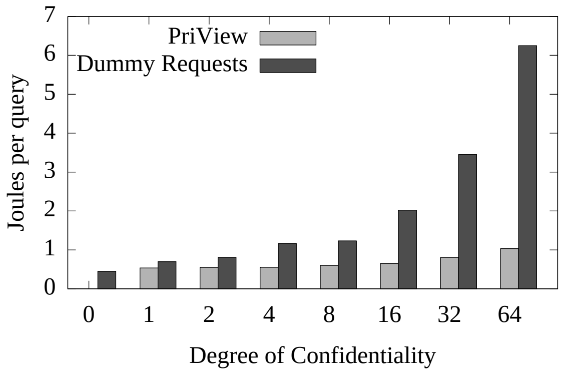

Fig. 4. **Power consumption increases sub-linearly for PriView as degree of confidentiality is increased.**

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 [11] 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.

V. 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. As future work, we plan on extending our implementation to support more than two proxies, evaluating the system with proxies running on different cloud providers, running against a wider range of services, and exploring techniques for improving the quality of dummy query generation. Ultimately, PriView's 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 *Proceedings of the 21st International Conference on Data Engineering Workshops*. IEEE Computer Society, 2005, p. 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*. 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] “P4400 Kill A Watt,” http://www.p3international.com/products/P4400.html.