Securely Outsourcing Cookies to the Cloud via Private Information Retrieval

Levon Nazaryan*, Ruofan Jin*, Chaoqun Yue*, Ozgur Oksuz*, Bing Wang*, Kyoungwon Suh†, Aggelos Kiayias‡
*Department of Computer Science & Engineering, University of Connecticut, Storrs, CT, USA
† Illinois State University, Normal, IL 61790, USA
‡ National and Kapodistrian University of Athens, 15784 Athens, Greece

***Abstract*—Many smartphone applications are web based and rely on cookies to maintain the status of a web session. Cookies, however, may lead to security threats since they may contain sensitive information. In addition, an attacker having access to a cookie can easily impersonate the legitimate user. In this paper, we propose and implement a system that securely outsources browser cookies to the cloud and ensures user privacy using Private Information Retrieval. Experimental evaluation using traces collected from operational cellular and WiFi networks demonstrates that our system achieves satisfactory performance for most real-life web browsing scenarios: the average latency is within 1.0 to 1.2 seconds (well within users' tolerance) even when retrieving tens of cookies over an LTE or WiFi network, and the amount of generated traffic is significantly lower than that when downloading the entire cookie database.**

***Keywords*-web cookies, cookie security, private information retrieval, cloud**

I. Introduction

Smartphones have become a prevalent computing platform. Their small physical forms make them particularly convenient computing devices to be used anytime anywhere, for a wide range of applications including surfing web, checking email, watching video, accessing social networking services, and online games. Many of these applications are web based. A widely used mechanism to keep the state of a web session is through cookies. Cookies may contain sensitive information, such as usernames, passwords, and credit card numbers [3]. In addition, unexpired cookies can be used even after restarting the web browser or the device. An attacker having access to a cookie can easily impersonate the legitimate user and continue the user's web sessions, thus leading to severe security threats.

The security threats posed by cookies have been well recognized (see [21] and the references within). Modern web browsers have implemented native protection mechanisms based on *Secure* and *HttpOnly* flags. These protection mechanisms however have only seen limited adoption [6]. In addition, while they have been designed to deal with various web and network attacks, they do not protect plaintext cookies that are stored on a device. Since smartphones are small devices that are more prone to loss and theft, plaintext cookies stored on smartphones are more vulnerable than those stored on laptops and desktops.

One way to protect cookies is not storing them locally, rather *outsourcing* them to secure external servers in the cloud. When a cookie is needed on a device, it is downloaded to the device, temporarily stored in the memory, and then removed from the memory after a short period of time (e.g., a few minutes or tens of minutes). In addition to security, this approach allows a user to conveniently switch among multiple platforms. Indeed, while relying on smartphones on the go, many people use different computing platforms depending on time and location. Outsourcing cookies allows the different platforms to share the cookies. As a result, a user can continue using a cookie when switching platforms, leading to seamless transition among the platforms.

Outsourcing cookies to the cloud, however, may not preserve user privacy as database servers may learn which cookies are requested. While encrypting the cookies provide confidentiality, it does not prevent statistical analysis attacks, which can be used to determine some (if not all) cookies (and hence the preferences of a user). To address this problem, we propose to use private information retrieval (PIR), which allows users to securely retrieve cookies from the servers while the servers never learn which cookies are requested. A trivial PIR technique is downloading the entire cookie database when any cookie is needed. This technique consumes a lot of network bandwidth. It is particularly problematic when the data are downloaded over a cellular network since users typically only have limited cellular data plan. In addition, downloading the entire cookie database may also lead to higher latency, affecting user experience.

In this paper we propose a system that allows users to securely outsource web browser cookies using state-of-the-art PIR techniques. To ensure privacy and confidentiality for the users, our system securely stores encrypted cookies in database servers in the cloud and uses a PIR protocol to retrieve cookies from the servers. The servers will send the requested cookies but will never know which cookies are requested.

Our main contributions are as follows:

- To the best of our knowledge, this is the first work that proposes outsourcing web browser cookies using PIR. This approach provides security while preserving user privacy. In addition, it allows cookies to be conveniently shared across multiple platforms.
- We design and implement a Firefox add-on that automatically requests and retrieves cookies required by a visited web page from the cloud using PIR.
- We use traces collected from operational cellular and WiFi networks to evaluate the performance of our system.

Evaluation results demonstrate that our system achieves satisfactory performance for most real-life web browsing scenarios: the average latency is within 1.0 to 1.2 seconds, well within users' tolerance [19], even when retrieving tens of cookies from the cloud over a cellular or WiFi network, and the amount of generated traffic is much lower than that when downloading the entire cookie database.

The rest of the paper is organized as follows. Section II briefly describes background. Section III describes the problem setting. Section IV describes the design and implementation of our system. Section V presents performance evaluation. Section VI reviews related work. Finally, Section VII concludes the paper.

II. Background

A. Web Cookies

A cookie is a small piece of data generated by a server, sent to and stored in a user's browser to maintain the state of a web session [3]. In this way, the server can track the user's multiple requests. A web browser cookie contains several fields: cookie name, cookie value (often a long string of random data), creation time, expiration date, path and domain that the cookie is valid for, and whether secure connection is needed (i.e., whether a cookie can only be sent over a secure connection, e.g., through SSL). Some cookies expire at the end of a session, e.g., those for online banking websites. For other cookies, the expiration date varies, often in days, months or years. Such cookies, when stolen, can be used to impersonate a legitimate user, causing severe security threats.

Our approach is outsourcing (encrypted) cookies to the cloud, and only downloading them when needed. The benefit of this approach is two-fold. First, it leads to better security. While a cookie may be stored in memory for a short period of them after being downloaded (e.g., for 10 minutes), the short storage period can make it more difficult for an attacker to steal the cookies. Second, it is convenient for a user to share cookies across multiple platforms. Cookie values (often times random strings) can be used across a wide range of platforms for maintaining the session status with a server. Therefore, smooth browsing experience can be enabled by downloading the relevant cookies when a user switches from one device to another. For instance, we have tested Alexa's top 10 websites, and verified that cookies for all these websites can be used across multiple platforms for smooth browsing.

B. PIR

PIR allows a user to retrieve an item out of multiple items from a database, while hiding from the database which item has been retrieved [17]. The database can be an electronic library, stock exchange share prices, pharmaceutical database, etc. In our context, the database stores web browser cookies.

The literature on PIR is extensive [15], [7], [17], [12]. Broadly, there are two types of PIR: information theoretic PIR and computational PIR. Information theoretic PIR provides absolute guarantee that each server participating in protocol execution gets no information about what users are after [31]. Computational PIR provides a weaker guarantee in that users have privacy against computationally bounded database servers. We consider information theoretic PIR in this paper since it provides better security guarantee.

One trivial form of PIR is that a database server sends the entire database to a client. For information theoretic PIR, Chor et al. [7] show that this trivial form of PIR is optimal when there is a single server. When there are multiple servers, each holding a copy of the database, they show that the communication overhead can be much lower. Therefore, we use multi-server information theoretic PIR in this paper.

Chor et al. propose a simple multi-server information theoretic PIR scheme that works over $GF(2)$ (binary arithmetic) [7]. This scheme, however, requires all the servers to respond to a query, and cannot deal with the situation when one server returns the wrong answer. In this paper, we use Goldberg's PIR scheme [12] that is more robust. Specifically, we use t-private k-out-of-ℓ PIR, which allows up to t servers to collude and it is sufficient when k out of the ℓ servers respond, $t < k \leq \ell$. For completeness, we briefly review Goldberg's PIR scheme. Before that, we briefly describe Shamir secret sharing, which is used in Goldberg's PIR scheme.

1) Shamir Secret Sharing: The classic Shamir secret sharing scheme [24] allows a dealer to choose a secret value σ, and distribute shares of that secret to ℓ players. If t or fewer of the players come together, they learn no information about σ. Specifically, let σ be an arbitrary element of some finite field $\mathbb{F}$ (not necessary uniformly distributed). The dealer selects ℓ arbitrary distinct non-zero indices $\alpha_1, \ldots, \alpha_\ell \in \mathbb{F}$, and selects t elements $a_1, \ldots, a_t \in \mathbb{F}$ uniformly at random. The dealer constructs a polynomial $f(x) = \sigma + a_1 x + a_2 x^2 + \ldots + a_t x^t$, and gives to player i the share $(\alpha_i, f(\alpha_i)) \in \mathbb{F} \times \mathbb{F}$ for $1 \leq i \leq \ell$. Note that the secret σ is just $f(0)$. Now any $t+1$ or more players can use Lagrange interpolation to reconstruct the polynomial f, and evaluate $f(0)$ to yield σ. However, t or fewer players learn absolutely no information about σ.

Sharing a vector of r elements in $\mathbb{F}^r$ (instead of a single field element) can be done in a straightforward manner: each coordinate of the vector is a secret shared separately, using r independent random polynomials.

2) Goldberg's PIR Scheme: Instead of working over $GF(2)$ as in the scheme by Chor et al. [7], Goldberg's PIR scheme works over a larger field $\mathbb{F}$, where each element can represent w bits ($w = \lfloor \log |\mathbb{F}| \rfloor$). The database D is an $r \times s$ matrix of elements of $\mathbb{F}$. In Goldberg's simplest construction, as with Chor's scheme, each server gets a copy of the database.

A client wishing to retrieve row β of the database generates a vector $\mathbf{e}_\beta$ that consists all zeros except for a single 1 in the coordinate β. To transform this into a t-private PIR protocol, the client uses (ℓ, t) Shamir secret sharing to share the vector $\mathbf{e}_\beta \in \mathbb{F}^r$ into ℓ independent shares $(\alpha_1, \mathbf{v}_1), \ldots, (\alpha_\ell, \mathbf{v}_\ell)$. That is, the client creates r random degree-t polynomials $f_1, \ldots, f_r$ satisfying $f_j(0) = \mathbf{e}_\beta[j]$ and chooses ℓ distinct non-zero elements $\alpha_i \in \mathbb{F}$. Server i's share will be the vector $\mathbf{v}_i = \langle f_1(\alpha_i), \ldots, f_r(\alpha_i) \rangle$. Each server then computes the product

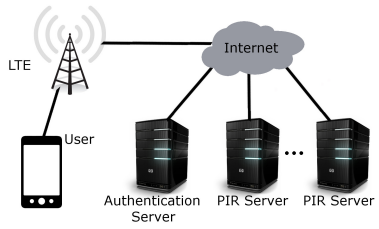


Fig. 1. Architecture of cookie outsourcing system.

$$\mathbf{r}_i = \mathbf{v}_i D = \langle \textstyle\sum_j f_j(\alpha_i)\,\mathbf{w}_{j1}, \ldots, \sum_j f_j(\alpha_i)\,\mathbf{w}_{js} \rangle \in \mathbb{F}^s.$$

$$\begin{pmatrix} f_1(\alpha_i) & \cdots & f_r(\alpha_i) \end{pmatrix} \begin{pmatrix} \mathbf{w}_{11} & \mathbf{w}_{12} & \cdots & \mathbf{w}_{1s} \\ \mathbf{w}_{21} & \mathbf{w}_{22} & \cdots & \mathbf{w}_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{w}_{r1} & \mathbf{w}_{r2} & \cdots & \mathbf{w}_{rs} \end{pmatrix} =$$

$$\begin{pmatrix} \sum_j f_j(\alpha_i)\,\mathbf{w}_{j1} & \cdots & \sum_j f_j(\alpha_i)\,\mathbf{w}_{js} \end{pmatrix}$$

By the linearity property of Shamir secret sharing, since $\{(\alpha_i, \mathbf{v}_i)\}_{i=1}^{\ell}$ is a set of Shamir secret shares of $\mathbf{e}_\beta$, $\{(\alpha_i, \mathbf{r}_i)\}_{i=1}^{\ell}$ is a set of Shamir secret shares of $\mathbf{e}_\beta D$, which is row β of the database. Goldberg's scheme further handles the case when only k out of the ℓ servers respond to a query, providing k-out-of-ℓ PIR.

III. Problem Setting

Fig. 1 presents the high level architecture of our system. A user outsources his cookies to multiple PIR servers. When the user browses a web page using a web browser and a cookie is needed, an add-on inside the web browser will contact the servers through a PIR protocol to retrieve the cookies. We next describe the security and attack models. The design and implementation of our system are deferred to Section IV.

A. Security Model

We assume $\ell \geq 2$ servers, and the PIR scheme is t-private k-out-of-ℓ PIR, $t < k \leq \ell$. That is, up to t servers can collude and it is sufficient when k out of the ℓ servers respond correctly to a query (the rest of the $\ell - k$ servers either do not respond because they are down or their data is not up to date). We assume all the servers are semi-honest, in that they follow the protocol specifications and do not change any data stored in users' database, do not change queries (from the web browser add-on), and do not change the query results, but they might try to learn extra information. This semi-honest server model is also used in other studies (e.g., [30], [22]) and is consistent with the cloud setting. The web browser add-on that communicates with the servers is fully trusted. We use the PIR protocol described in Section II-B2.

B. Attack Model

In our system, once a cookie is downloaded, decrypted and stored temporarily in memory in a device, it suffers the same types of attacks for web cookies. The attacks can be web attacks (e.g., cross-site scripting), network attacks (e.g., an attacker can intercept a plaintext cookie transmitted over the network) or end-system attacks (e.g., an attacker copies a plaintext cookie from the memory or hard disk of a device) [21], [6]. Cookies stored on smartphones are even more vulnerable because malicious apps can be downloaded from official application stores [28], which can access a user's private files stored on the device and sends them to an attacker. In addition, smartphones are more prone to theft and loss because their small form factors, making them more vulnerable to end-system attacks. As an example, an attacker, once gets hold of a smartphone, can uses special hardware/tools to access a device's memory and read confidential files or information stored in the memory.

Our approach of outsourcing cookies is complementary to other approaches in making cookies more secure (see related work in Section VI). In particular, since our system outsources encrypted cookies to external servers and only stores cookies temporarily in memory for a short period of time (a tunable parameter), it makes end-system attacks more difficult.

IV. System Design and Implementation

In this section, we first describe the design of our system to show what entities are involved and how they interact to each other. At the end, we present our system implementation.

A. Design Overview

Fig. 2 shows the design of our system. It consists of two parts: the client and the PIR servers. Each server stores an exact copy of cookie database in the encrypted form. The client is a web browser. It contains a web browser add-on, which contains an *index database*, a *PIR query generator*, and a *PIR reply decoder*. Each server contains the same encrypted cookie database, where an encrypted cookie is stored as a row in the database.

When a user visits a web page that requests a cookie, if the cookie is not stored in the web browser, the browser add-on checks the index database (along with other parameters) to find the index of the cookie (e.g., row β of the database), and passes it to the query generator. The query generator generates queries to request the cookie from the multiple servers using the PIR query generation scheme. When a server receives the corresponding query, it computes the response and sends the reply to the PIR reply decoder in the browser add-on, which performs additional computations over the responses from the multiple servers to recover the encrypted cookie. Finally, the browser add-on decrypts the cookie, and passes it to the accessed web page.

The above description assumes that a user has already authenticates himself to an authentication server. Only after the authentication, the web browser add-on can send queries to the PIR servers. We next describe the authentication process and cookie encryption in more detail.

B. Authentication and Confidentiality

A user authenticates himself to an authentication server. The authentication can be through password or other means. To

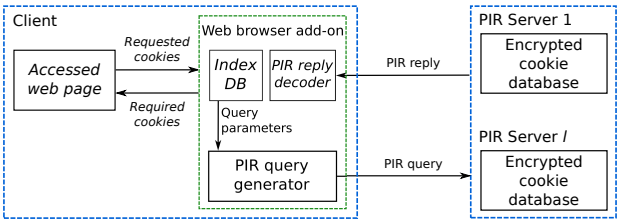


Fig. 2. The system architecture.

protect our system against brute force attacks, a standard rate-limiting mechanism can be enabled for failed authentication attempts (e.g., the system waits for 30 seconds before accepting new authentication attempts after 5 consecutive failed attempts). Once authenticated, the user will be able to use the add-on for a limited amount of time, T, referred to as *threshold*. Specifically, within time T, any new request for a cookie will be directed to the servers; after time T, the user will be asked to authenticate himself again if a cookie is needed. The value of T presents a tradeoff: shorter T is more secure but inconvenient; longer T is less secure but convenient. It can be set by a user based on his preference. It can also be set automatically based on a user's browsing characteristics and time of the day. For instance, if a user typically checks emails and social network accounts every morning (e.g., 8-10am) for around half an hour, then T can be set automatically to half an hour for the morning; it can be set to a lower value in the afternoon when a user is busy with work and does not access the Internet often. When cookie i is downloaded, the amount of time that the cookie is kept in memory, T_i, can also be a tunable parameter. One way is to set $T_i = T$. This, however, does not account for the heterogeneity of the websites since some websites may be accessed more often than others. Therefore T_i can be different for different websites, which can also be set automatically based on user browsing behavior. In our evaluation (Section V), we simply set $T_i = T, \forall i$ and vary T from 5 to 60 minutes. A detailed study on setting T and T_i and their impact on performance is beyond the scope of the paper and is left as future work.

For confidentiality, a user needs to encrypt cookies before outsourcing them to the cloud (the same key is also used to decrypt the retrieved cookies later on). Specifically, we use AES-256 in Cipher Block Chaining (CBC) mode and a random Initialization Vector (IV) to encrypt cookies. The IV and the ciphertext are stored in the cookie database (see Section IV-C). When a user authenticates himself, the key gets decrypted (e.g., the key is encrypted using the user's password, and it is decrypted after the user enters the password for authentication). After time T, the key is encrypted again automatically. As mentioned earlier, while encrypting the cookies provides confidentiality, it does not prevent statistical analysis attacks at the servers. Therefore, we use PIR, which allows users to securely retrieve cookies from the servers while the servers never learn which cookies are requested.

Confidentiality can also be achieved by encrypting and storing the entire cookie database locally at a client with a key; the database is decrypted when needed by asking user to enter a password, and then encrypted after time T. Our scheme is better than the above solution in two important aspects. First, our system only keeps a small number of decrypted cookies (instead of the entire cookie database) temporarily in memory. Secondly, as mentioned earlier, since the cookie database is stored in the cloud, it allows convenient cookie sharing and hence smooth browsing over multiple platforms.

C. Main Operations

We next describe the main operations in our system, including creating cookie database, retrieving, updating and removing cookies. The cookie database stores a set of encrypted cookies. Since cookies can be of different lengths, we pad each cookie so that they are of the same length, and then encrypt the cookies. Each encrypted cookie is stored as a row (also called a block) in the cookie database. Let r denote the number of rows in the database, let s denote the number of words in a block, and let w denote the number of bits in a word. Then each block (or encrypted cookie) contains $s \times w$ bits. Suppose the size of the database is measured in words. A square shape database requires r to be equal to s, which may require adding extra blocks. Each added block contains a sequence of random bits so that it is not differentiable from the encrypted cookies. The encrypted cookie database, once constructed, will be sent to each of the PIR servers, so that each server has a copy.

An index database stores indexes (or locations) of the cookies in the outsourced database. For a given website, e.g., google.com, our system searches the index database to find the index of each cookie that is associated with this website. Suppose a cookie is in row β according to the index database. Our system then uses the index (i.e., β) and additional parameters (see Section II-B) to construct queries for each PIR server, send the queries to the PIR servers, and uses the responses from the servers to recover the encrypted row of the database. The encrypted cookie is then decrypted. Since a cookie may be padded with extra bits, the index database also stores for each cookie, the number of bits in the cookie. After a cookie (with padded bits) is decrypted, this information is used to striped out the extra bits.

If the user visits a website for the first time and the website tries to create cookies (or if a website tries to update existing cookies), then both the index database and the cookie database need to be updated. To update an existing cookie in the cookie database, the cookie's index in the index database has to be found. Suppose its index is β. Then the new encrypted cookie is transmitted to each of the PIR servers, and overwrites the βth row in the cookie database at each server. When a cookie needs to be added to the cookie database, our system first finds an index that has not been used yet, assigns the index to be the index of the cookie, saves the index into the index database, and outsources the encrypted cookie to overwrite the corresponding row in the cookie database in each PIR server.

To remove an existing cookie from the cookie database, this cookie's information is removed from the index database. As a

result, the index for the cookie immediately becomes available for use again. The corresponding row for the cookie will be replaced with random bits.

D. System Implementation

Our system implementation leverages a publicly available PIR implementation called Percy++ [12], which is written in C++ and implements the PIR protocols in [12], [10], [18], [9]. In the original Percy++, the client requests and retrieves cookies from the multiple servers in *sequence*. We modified the source code of Percy++ so that the client sends requests to and receives responses from multiple servers in *parallel*, which can significantly reduce latency, particularly for networks with long latency (e.g., cellular networks).

The browser add-on is implemented in Firefox[1], on top of the PIR client in Percy++. We choose Firefox because Firefox is free, open-source and has powerful API and a large community support. Firefox stores cookies in an SQLite format database called *cookies.sqlite* in the user's active profile directory. To construct a cookie database for Percy++, we extract all cookies from the *cookies.sqlite* database using *nsICookieManager* [5]. The cookies are padded and encrypted using the AES-256 algorithm with the user's encryption key. The encrypted cookies are then stored in a Percy++ database.

The client and servers run in the same finite field, specifically we use $GF(2^8)$, which has been shown to be much more efficient than Percy++'s default mode (arithmetic in $Z \mod p$) [12]. For example, in $GF(2^8)$ mode, it takes 0.7 seconds while the default mode takes more than 2.5 seconds to retrieve 15 cookies from a $574,564$ byte database when using a trace collected from a LTE network (see more details on the trace in Section V. The retrieved cookies (when they are decrypted and their paddings are removed) are passed to the browser add-on, which imports them to Firefox using *nsICookieManager*. Firefox uses the cookies and, if the retrieved cookies are unexpired and contain valid information for the website, then the user logs in to his/her online account.

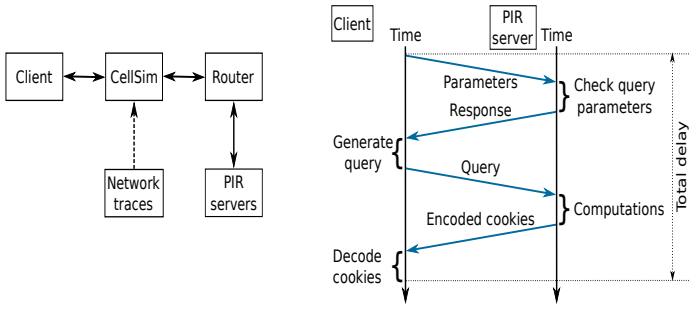
V. Performance Evaluation

In this section, we evaluate the performance of our system. The evaluation focuses on cookie retrieval because it is used much more often than the other two functions (i.e., creating cookie database and updating cookies). In addition, the performance of cookies retrieval is critical to user experience. In the following, we first describe the experimental setup, and then detail the evaluation results.

A. Experimental Setup

We use a cookie database of size $r \times s$ words, each word being 8 bits. Specifically, $r = s = 758$. That is, it can store up to 758 cookies, each of at most 758 words. The size of the database is $4,596,512$ bits, approximately the same as the default cookie database size of $4,194,304$ bits

[1]We used Firefox version 30.0 in both our implementation and experiments. It is also possible to implement the same add-on in other modern web browsers.



(a) Experimental setup. (b) Timing diagram.

Fig. 3. Experimental setup (a) and timing diagram (b) when retrieving cookies.

in Firefox. At the client, the Firebox add-on fetches cookies from the servers. We investigate two scenarios: one with two PIR servers, both are honest (i.e., they respond to queries with the correct answers) and do not collude with each other, the other with three PIR servers, and one server does not respond to queries. In the interests of space, we only present the results for the case with two PIR servers; the results for the case with three servers are similar.

We conduct experiments on three virtual machines (VMs). One VM is for the client and the other two VMs are for the PIR servers. Each VM has 2 GB RAM and a dual-core 2.4 GHz Intel Core i5 CPU running Ubuntu 12.04 (32bit). The VMs are configured to be of comparable capabilities of a modern smartphone. Note that increasing the RAM does not speed up the computations on PIR servers as the servers already have enough RAM to run Ubuntu and to load the entire database into RAM to make query processing faster. In practice, a server may be more powerful than the configuration that we use, and hence can achieve even better performance than what we report.

Fig. 3(a) shows the setup of our experiments. The two PIR servers are connected to a local network, which is further connected to the client. To evaluate the performance of our system in a repeatable environment, we exploit a trace-driven simulation environment, CellSim [29], that uses traces collected from operational networks to simulate the network between the client and the servers. CellSim serves as a transparent Ethernet bridge that schedules packet transmission according to the pre-collected traces. Specifically, it runs on a PC, takes in packets on two Ethernet interfaces, delays them for a configurable amount of time (the propagation delay), and adds them to the tail of a queue. It releases packets from the head of the queue to the other network interface according to the network trace that is being played back.

We consider both cellular and WiFi networks, the two primary types of access networks that a smartphone connects to. Specifically, we collect a 13.2 minutes long trace from a commercial 4G LTE network and a 15.7 minutes long trace from a research university's campus WiFi network. On average, the LTE network's trace file has 13.2 Mbps downlink and 2.4 Mbps uplink bandwidth (bitrate), and the WiFi network's trace file has 13.9 Mbps downlink and 11.6 Mbps uplink

TABLE I
THE AVERAGE DOWNLINK AND UPLINK BANDWIDTH OF THE COLLECTED NETWORK TRACES.

Network	Downlink	Uplink
WiFi	13.9 Mbps	11.6 Mbps
LTE	13.2 Mbps	2.4 Mbps

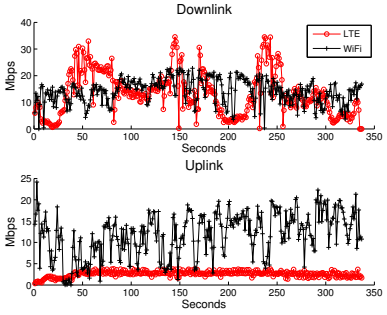


Fig. 4. Uplink and downlink bandwidth versus time for LTE and WiFi traces.

bandwidth. Table I shows the average link bandwidth (or bitrate) in both downlink and uplink directions of the collected traces. Fig. 4 plots the bandwidth versus time (only the first 350 seconds of the traces are being plotted). For the downlink direction, the bandwidth of the LTE trace exhibits much more rapid changes over time than that of the WiFi trace. For the uplink direction, the average bandwidth of the WiFi network trace is significantly larger than that of the LTE network trace.

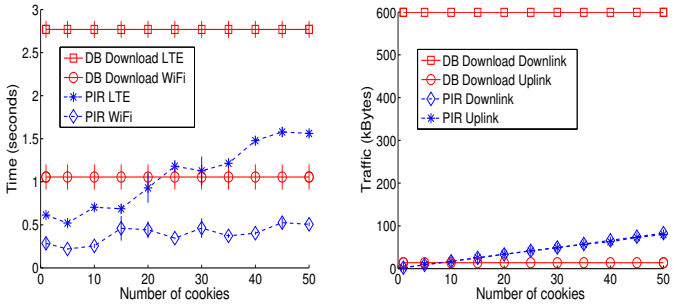
We compare the performance of our system with the trivial scheme that downloads the entire cookie database from one server upon a request. For this trivial scheme, the network connecting the client and the server is also controlled by CellSim. Specifically, we compare these two schemes under the same network setting by feeding the same network trace to CellSim. In all the settings, the propagation delay between the client and each server is set to 40 ms.

B. Computation and Communication Overhead

The latency for a client to retrieve cookies from one server include the following components: (1) the time to generate a query; (2) the time to send the query to the server; (3) the time for the server to compute responses; (4) the time for the server to send the response back to the client; and (5) the time for the client to decode the response to recover cookies. Fig. 3(b) shows the timing diagram. Note that the requests to the two servers and the responses from the servers are in parallel.

We first report computation latency in our system. Experimental results show that encrypting and decrypting 50 cookies using AES-256 takes 5.83 and 5.35 ms respectively. The PIR computation overhead increases roughly linearly with the number of cookies that are requested. For a small number of cookies, the computation latency is only a few milliseconds. Even when the number of cookies is 50, the computation latency is only 35.7 ms. Overall, the computation latency is very low.

We next present the total latency under the LTE and WiFi network settings. Since the computation latency is negligible,



(a) Latency. (b) Generated traffic.

Fig. 5. Total latency and the amount of generated traffic (the aggregate traffic between the client and the two servers) when retrieving different number of cookies.

most of the latency is due to communication delay. For each setting, we make 100 simulation runs sequentially in CellSim, which use different portion of the network trace, to obtain the average results and the 95% confidence intervals. Fig. 5(a) plots the results when the number of requested cookies varies from 1 to 50. The results when using PIR and when downloading the entire cookie database are both shown in the figure. We observe that for both LTE and WiFi networks, retrieving even 50 cookies using PIR is much faster than downloading the entire cookie database. Specifically, when using PIR, all the latencies are below 1.6 seconds.

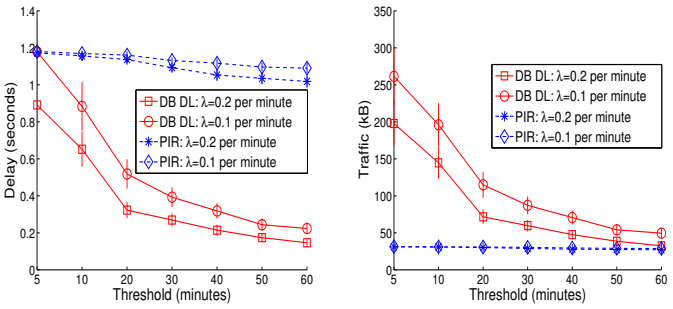
Fig. 5(b) plots the number of bytes that are transferred in the network for both the downlink and uplink directions (the result is independent of the network that is being used). When using PIR, the amount of data along the two directions is approximately the same [12], and increases linearly with the number of cookies. When downloading the entire cookie database from one server, the amount of data in the downlink direction is around 600 KB (approximately the size of the cookie database with additional headers); the amount of data in the uplink direction is much lower (they are mainly TCP ACKs). The amount of traffic under the PIR scheme with two servers is significantly lower than that when downloading the entire cookie database from one server for the downlink direction (even when requesting 50 cookies).

C. Energy Consumption

The total energy consumption at the client is the sum of energy used by the CPU for computation and by the radio interface for data transfer. Since both types of energy consumption increases with the number of cookies that are requested, we next only consider requesting 50 cookies. Our experiments show that the PIR computation overhead for 50 cookies is 35.7 ms at about 1.0% CPU utilization. Decryption of the 50 cookies takes 5.35 ms at 2.9% CPU utilization. CPU power consumption is mainly influenced by CPU utilization and frequency [32]. The results in [27] show that the CPU energy consumption coefficient at different CPU utilizations is below 4.34 mW/%. Hence, to retrieve and decrypt 50 cookies the client phone's CPU will require 0.22 mJ energy. When not using PIR, decrypting the entire cookie database takes 16.4 ms

TABLE II
NUMBER OF COOKIES FOR TOP 10 ALEXA.COM WEBSITES (AS OF MAY 31, 2016).

Rank	Web Page	Number of Cookies	
		Not Logged-In	Logged-In
1	Google.com	3	17
2	Youtube.com	3	21
3	Facebook.com	4	15
4	Baidu.com	7	13
5	Yahoo.com	5	13
6	Wikipedia.org	5	47
7	Amazon.com	7	12
8	Twitter.com	6	12
9	QQ.COM	11	36
10	Live.com	3	6



(a) Average delay. (b) Average generated traffic.

Fig. 6. Average delay and generated traffic (aggregate traffic between the client and the two servers) per visited website when using LTE network, $N = 100$, $\alpha = 0.7$.

at 2.5% CPU utilization. Therefore, the energy consumption is 0.18 mJ, slightly lower than that using PIR.

For 50 cookies the client uploads 79.9 KB and downloads 83.0 KB data when PIR is used (see Fig. 5 (a)). When downloading the entire cookie database, the client sends 13.8 KB and receives 599.5 KB data. Measurements in [13] show that LTE requires 1680.2 mW power and WiFi requires 124.4 mW power when in full power mode, much more than CPU power consumption. Using the average downlink and uplink bandwidth for WiFi or LTE network (see Section V-A), we calculate the average amount of time the corresponding radio interface has to be in full power mode to transfer data. In particular, when PIR is used, WiFi radio has be in full power mode for 47.8 ms to download data and for 55.1 ms to upload data (50.3 ms and 266.2 ms to download and upload data under LTE respectively). When downloading the entire database, WiFi radio has to be in full power mode for 345.0 ms to download data and 9.5 ms to upload data (363.3 ms and 46.1 ms to download and upload the data in LTE respectively). Clearly, when downloading the entire cookie database, network interfaces have to be in full power mode longer than when PIR is used (approximately 3.5 times longer for WiFi and 1.3 times longer for LTE), leading to significantly more energy consumption.

D. Multiple Web Sessions

For a large scale evaluation where a user issues multiple web requests during a time period, we consider a WiFi access point (AP) association trace, which contains the AP association information for about $53,807$ unique users during 9am to 6pm in a day. We randomly select the information for 20 users. For each user, we have a sequence of time points when a user connects to an AP and the duration of the connection. The time series is then used to represent when a user is connected and for how long he is connected to the Internet (in the following, we assume the connection can be either through LTE or WiFi).

We further need to simulate when and what websites a user visits while the user is connected to the AP. For this purpose, we assume that the websites are taken from the top N websites from Alexa, and the requests for a website follows a Zipf distribution [11]. Specifically, for the i^{th} most popular website in Alexa, the probability that the website is accessed is proportional to $1/i^{\alpha}$, where $\alpha \geq 0$ is a constant. For the top 10 Alexa websites, we record the number of cookies that each website requires (in two cases, when the user is logged in and not logged in), see Table II. All websites listed in the table do not have third-party cookies and most of their cookies do not expire for at least 1 week. Hence these cookies can be outsourced to the cloud using PIR. If a user visits one of the top 10 Alexa websites, then the number of cookies that is requested is set to the number of cookies required by the website (using the value in the logged-in case). For all other websites, for simplicity, we assume the required number of cookies is 20, which is the average of the number of cookies of the top 10 Alexa websites. Last, we assume that the average interval between two consecutive website requests is $1/\lambda$, where λ is the rate of requesting websites.

We next present the evaluation results. The threshold, T, is varied from 5 to 60 minutes. The performance metrics are the average latency and average amount of generated traffic for servicing a web request. For a web request whose cookies are already stored locally, the latency and the amount of traffic are both zero. We set $N = 100$ or 500, $\alpha = 0.7$ or 1.0, and $\lambda = 0.1$ or 0.2 request per minute (i.e., on average, a user sends a request every 5 or 10 minutes). In the following, we only present the results for one setting in the LTE network; the results in other settings have similar trend.

Fig. 6(a) plots the average latency (along with 95% confidence intervals) per website visit when using the LTE network. The average is obtained by simulating the web access patterns of 20 users during a day (9am to 6pm) in a real-world scenario (the simulation is repeated 100 times by generating web requests following the process described earlier). The threshold T is varied from 5 to 60 minutes, $N = 100$ and $\alpha = 0.7$. The results for $\lambda = 0.1$ and 0.2 request per minute are both plotted in the figure. When using the PIR scheme, the average latency is 1.0 to 1.2 seconds (the confidence intervals are tight), which is well within users' tolerance [19]. When downloading the entire cookie database, the latency decreases quickly when the threshold increases. For increased security, however, it is better to keep the threshold reasonably short (e.g., a few minutes or less). Fig. 6(b) plots the average amount of generated traffic per visited website. The average amount of traffic per visited website is about 30 to 40 KB when using the

PIR scheme, which is lower than that when downloading the entire cookie database from one server, except for very large threshold values. As mentioned before, using large threshold values is however not desirable due to decreased security.

VI. Related Work

Many studies have proposed techniques to prevent or mitigate attacks to web cookies [8], [14], [23], [20], [26], [25]. Modern native web browser vendors developed *Secure* and *HttpOnly* flags. The HttpOnly flag makes a cookie only accessible upon transmission of HTTP(S) requests, thus blocks accesses by non-HTTP APIs and is effective in preventing code injection attacks (such as cross-site scripting). The Secure flag ensures that a cookie is transmitted only over secure communication channels. A recent study [6] assesses the robustness of these two flags against both web and network attacks. The authors further develop *CookiExt* that provides client-side protection against the theft of session cookies, based on appropriate flagging of such cookies and automatic redirection over HTTPS requests carrying such cookies. Existing studies also develop secure cookies that provide secure services such as authentication, confidentiality, integrity and anti-replay (e.g., [21], [16]).

Our work differs from the above studies in that we outsource cookies to external secure servers using PIR. Not storing cookies locally is particularly useful in defending end-system attacks. In addition, it is convenient for a user to share cookies across multiple platforms. On the other hand, our work can be combined with existing approaches. For example, once a cookie is downloaded and decrypted, Secure and HttpOnly flags can be used when transmitting the cookie to the web server.

Our work also differs from various synchronization tools [4], [1], [2] that let users store and synchronize different type of files (e.g., documents, pictures) across devices. It also differs from bookmark synchronizing tools that are built in modern web browsers.

VII. Conclusion

In this paper we have proposed a system that securely outsources web browser cookies using PIR. We presented the system design and an implementation as a Firefox add-on. Our experiment results show that the system achieves satisfactory performance for most real-life web browsing scenarios. In particular, our system transfers significantly less data between the client and servers compared to downloading the entire cookie database. In addition, the average latency is within 1.0 to 1.2 seconds even when retrieving tens of cookies from the cloud over LTE or WiFi network, well acceptable for most users browsing the Internet.

References

[1] Dropbox. dropbox.com. [Online; accessed Sep. 1, 2016].

[2] Google Drive. drive.google.com. [Online; accessed Sep. 1, 2016].

[3] HTTP cookie. wikipedia.org/wiki/HTTP_cookie. [Online; accessed Sep. 1, 2016].

[4] iCloud. apple.com/icloud. [Online; accessed Sep. 1, 2016].

[5] Mozilla Developer Network. developer.mozilla.org. [Online; accessed Sep. 1, 2016].

[6] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan. Automatic and robust client-side protection for cookie-based sessions. In *Proc. of ESSoS*, 2014.

[7] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6), Nov 1998.

[8] I. Dacosta, S. Chakradeo, M. Ahamad, and P. Traynor. One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. *ACM Transactions on Internet Technology*, 12(1), 2012.

[9] C. Devet and I. Goldberg. The best of both worlds: Combining information-theoretic and computational PIR for communication efficiency. In *Proc. of PETS*, 2014.

[10] C. Devet, I. Goldberg, and N. Heninger. Optimally robust private information retrieval. In *Proc. of the 21st USENIX Conference on Security Symposium*, 2012.

[11] X. Gabaix. Zipf's law for cities: An explanation. *Quarterly Journal of Economics*, 114, 1999.

[12] I. Goldberg. Improving the robustness of private information retrieval. In *Proc. of IEEE Symposium on Security and Privacy*, 2007.

[13] J. Huang, F. Qian, A. Gerber, M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *Proc. of MobiSys*, 2012.

[14] E. Kirda, C. Krugel, G. Vigna, and N. Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *Proc. of SAC*, 2006.

[15] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: single database, computationally-private information retrieval. In *Proc. of FOCS*, 1997.

[16] A. X. Liu, J. M. Kovacs, and M. G. Gouda. A secure cookie scheme. *Computer Networks*, 56(6), 2012.

[17] C. A. Melchor. High-speed single-database PIR implementation. In *Proc. of PETS*, 2008.

[18] C. A. Melchor and P. Gaborit. A lattice-based computationally-efficient private information retrieval protocol. Cryptology ePrint Archive, Report 2007/446, 2007.

[19] F. F.-H. Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3), 2004.

[20] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen. SessionShield: lightweight protection against session hijacking. In *Proc. of ESSoS*, 2011.

[21] J. S. Park and R. Sandhu. Secure cookies on the web. *IEEE Internet Computing*, 4(4), July 2000.

[22] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proc. of SOSP*, 2011.

[23] P. D. Ryck, L. Desmet, W. Joosen, and F. Piessens. Automatic and precise clientside protection against CSRF attacks. In *Proc. of ESORICS*, 2011.

[24] A. Shamir. How to share a secret. *Commun. ACM*, 1979.

[25] U. Shankar and C. Karlof. Doppelganger: Better browser privacy without the bother. In *Proc. of CCS*, 2006.

[26] S. Tang, N. Dautenhahn, and S. T. King. Fortifying web-based applications automatically. In *Proc. of CCS*, 2011.

[27] N. Vallina-Rodriguez and J. Crowcroft. Energy management techniques in modern mobile handsets. In *IEEE Communications Surveys and Tutorials*, 2012.

[28] T. Wang, K. Lu, L. Lu, S. Chung, and W. Lee. Jekyll on iOS: When benign apps become evil. In *Proc. of USENIX Security*, 2013.

[29] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Proc. of NSDI*, 2013.

[30] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving queries on encrypted data. In *Proc. of ESORICS*, 2006.

[31] S. Yekhanin. Private information retrieval. *Communications of the ACM*, 53(4), April 2010.

[32] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of IEEE/ACM/IFIP International Conference on Hardware/Software-Codesign and System Synthesis*, 2010.