Privad: Rearchitecting Online Advertising for Privacy

Technical Report: MPI-SWS-2009-004

Saikat Guha, Bin Cheng, Alexey Reznichenko, Hamed Haddadi, Paul Francis
Max Planck Institute for Software Systems
Kaiserslautern-Saarbrücken
Germany
{sguha,bcheng,areznich,hamed,francis}@mpi-sws.org

Oct. 2, 2009

Abstract

This technical report describes an architecture and protocols for personalized online advertising system called Privad that is far more private than current systems exemplified by Google and Facebook. This report also provides an analysis of the privacy characteristics of Privad.

Contents

1 Introduction

Online advertising is a key economic driver in the Internet economy. It funds services provided by such industry giants as Google and Facebook, and helps pay for data centers and, indirectly, ISPs. Internet advertisers increasingly work to provide more personalized advertising. Unfortunately, personalized online advertising, at least so far, has come at the price of individual privacy [24] and poor user experience [26]. And while privacy advocates would like to put an end to advertising models that violate privacy, aside from a few highly publicized battles [20], they have had little success with the more entrenched ad brokers like Google and Yahoo! [14]. Arguably the reason why privacy advocates have failed is that they offer no viable alternatives. The deal they offer, privacy *or* personalization [3, 17, 34], is not acceptable to the entrenched players. This report is a proposal for that alternative.

This report presents a practical private online advertising system, which we call Privad. Our high-level goals for Privad are that it:

1. is private enough, and certainly substantially more private than current systems,
2. is at least as scalable as current systems,
3. targets ads as well as or better than current systems, and
4. fits within the current business framework for online advertising.

As these goals suggest, we are looking for a design that finds a sweet spot between privacy and other practical aspects of the system (scalability, targeting, business model). A key question is, how private is private enough? Current advertising systems, such as Google and Yahoo!, are in a deep architectural sense not private: they gather information about users and store it within their data centers. Users are forced to simply trust them not to do anything bad with the information. For example, there is little to stop an employee of one of these advertising companies from determining who is likely to have certain health problems, and selling this information to insurance companies [23].

We argue that Privad is considerably more private than current systems. Privad does not, for instance, require trust in any single organization. But is it private enough? There is obviously no single universal answer to this question. We believe that ultimately it is up to society to decide what is private enough, and society here tends to be represented by consumer and privacy advocacy groups like the Electronic Frontier Foundation (EFF), the American Civil Liberties Union (ACLU), and others [5]. Our strategy, then, is to

design and build a system that is as private as possible while still achieving the practical goals, and to then see who believes that it is private enough.

The core idea behind Privad is simple: the user's profile is kept on the user's computer. Deciding what ads to show, as well as serving the ads, is performed purely locally by the user's computer. This is made possible by pushing a mix of ads to users in advance. Reports about which ads are viewed or clicked are transmitted in such a way that user privacy is preserved while still allowing the advertising network to detect and defend against click-fraud.

Privad has several benefits over the current advertising model. First, Privad is verifiably private. Not only is the Privad protocol private by construction, it admits third-party verifiers that can ascertain that parties do not deviate from the protocol. This is key to convincing privacy advocates to endorse Privad, which we believe will be crucial for deployment. Second, serving ads locally eliminates multiple round-trips to the ad servers before the webpage can be rendered. Thus, as one might expect, Privad results in a more interactive browsing experience while reducing the load on the ad servers. And third, we argue that Privad has the potential for better targeting than online advertising today. This is in part because Privad has access to substantially more information, and in part because of privacy guarantees that should increase the level of personalization that users and privacy watchdogs can accept. In improving ad relevance, Privad continues the direction set by Google that has proven that users value more relevant ads.

One key challenge is incentivizing deployment. Privad is not aimed for users that today disable ads altogether. For users that do view, and occasionally click ads today, deploying requires first that Privad not degrade user experience in any way. We can ensure this by only showing ads in the same ad boxes that are common today (unlike previous adware, which employed disruptive advertising). Second, especially early on there must be some positive incentive for users to install it. This could be done through bundling other useful software, shopping discounts, or other incentives. Finally, it requires that privacy advocates (e.g. EFF, ACLU, and government agencies) endorse Privad. This at least prevents anti-virus software from actively removing Privad from clients. Ideally, it even leads to privacy-conscious browser vendors (e.g. Firefox) or operating systems installing it by default, or by governments mandating that existing advertising companies deploy Privad technology.

2 Privad Model

There are five players in Privad: user, publisher, advertiser, broker, and dealer. User, publisher, advertiser, and broker all have analogs in today's advertising model, and play the same basic economic roles. Users visit publisher webpages. Advertisers wish their ads to be shown to users on those webpages. The *broker* (e.g. Google) brings together advertisers, publishers, and users. For each ad viewed or clicked, the advertiser pays the broker, and the broker pays the publisher.

There are two key components to privacy in Privad. First, the task of profiling the user is done at the user's computer rather than at the broker. This is done by *client* software running on the user's computer. Second, all communication between the client and the broker is proxied anonymously by the *dealer*. The dealer is run by an organization that is itself untrusted with user profile information, but is nevertheless unlikely to collude with the broker. This could for instance be prominent privacy advocacy groups (e.g. EFF or ACLU) or a government regulatory agency. The dealer's operational costs could be covered by a special tax levy on the broker.

The dealer serves two roles. For the user the dealer provides anonymity by hiding the user's identity (e.g. IP address) from the broker, but itself does not learn any profile information about the user since all messages between the client and broker are encrypted. Unfortunately, when clients are hidden from the broker, the broker is less able to protect itself against click-fraud. Therefore, the dealer also helps the broker defend against click-fraud, but in a way that preserves user privacy. Additionally, the dealer helps protect against application-level DoS at the broker by rate-limiting client messages.

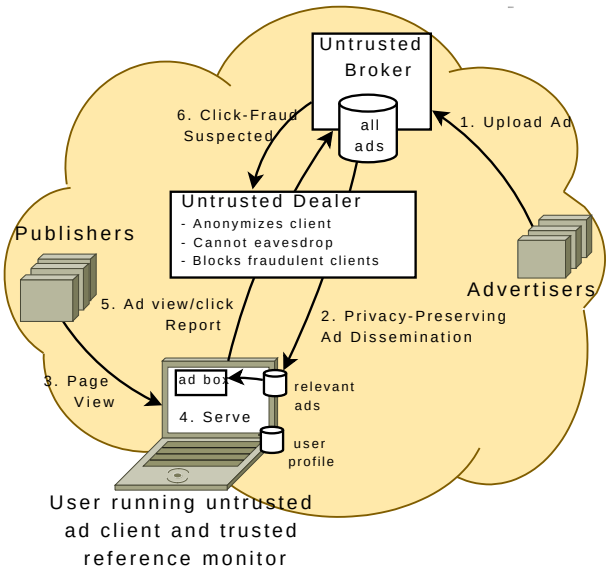


Figure 1: The Privad architecture

3 Privad Overview

3.1 Message exchanges

Figure 1 illustrates an overview of Privad operation. The client monitors user activity (for instance webpages seen by the user, personal information the user inputs into social networking sites, the contents of emails or chats sessions, and so on) and profiles both the *interests* and *demographics* of the user. Interests include products or services like sports.equipment.tennis or outdoor.lawn-care. Demographics include things like gender, age, salary, and location.

Advertisers upload ads to the broker, including the set of interests and demographics targeted by each ad. The broker distributes some fraction of those ads to clients through the dealer. This is done through a pub-sub mechanism whereby the client subscribes to a set of ads corresponding to a single interest category and a few broad demographics like gender geographic area, and language, and the broker delivers those ads. All messages between client and broker are made anonymous by the dealer, and are encrypted so that the dealer does not see their contents. If the user has multiple interests, there is a separate subscription for each interest, and the broker cannot correlate the separate subscriptions to the same user. Note that this distribution does not take into account the full set of demographics of the user. As a result, the client receives both ads that are and are not targeted to the user.

When the user browses a website that provides ad space, or runs an application like a game that includes ad space, the client selects an ad and displays it in the ad space. A report of this *view* is anonymously transmitted to the broker via the dealer. If the user clicks on the ad, a report of this *click* is likewise anonymously transmitted to the broker. The broker uses these reports to bill advertisers and pay publishers. The broker also forwards the reports (or summaries) to the advertisers so that they may better manage their ad campaigns.

Unscrupulous users or clients may launch click-fraud attacks on publishers, advertisers, or brokers. Both the broker and dealer are involved in detecting and mitigating these attacks. The goal of mitigation is for the dealer to identify attacking clients and suppress their reports. There are two mechanisms for identifying attacking clients. First, the dealer may identify an attacking client directly when the client transmits too many reports or subscription requests. Second, the broker identify which publishers or advertisers are under

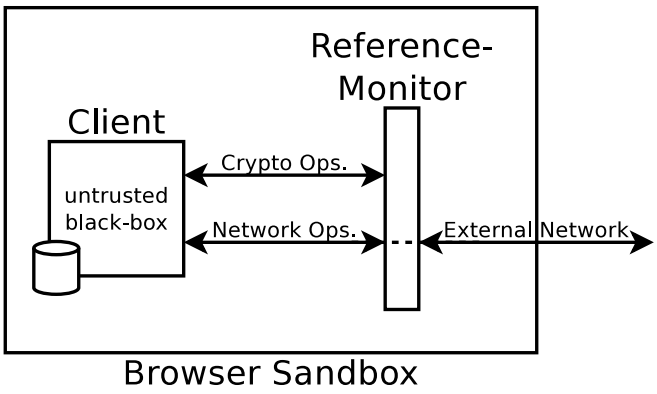


Figure 2: The client architecture

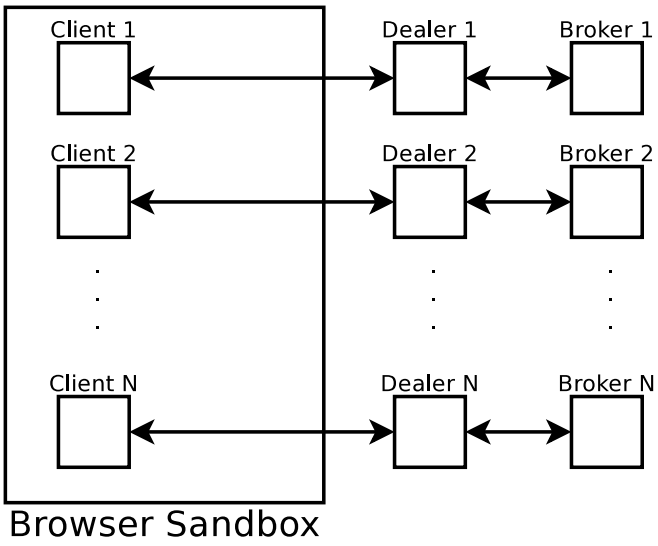


Figure 3: Multiple independent clients

attack, and indicate to the dealer which reports or subscriptions relate to these publishers or advertisers. The dealer then associates these reports with clients. Any clients associated with a threshold number of attacks are identified as attackers.

3.2 Client Architecture

Messages between client and broker are encypted so that the dealer cannot see their contents. However, it is critical that the user, or privacy advocates operating on behalf of users, are confident that no private information is being covertly transmitted in the encrypted message. Towards this end, the Privad client architecture allows for a reference moniter (Figure 2). The reference monitor allows any third party software to intercept all messages between the client and the network, and to execute the crypto operations. This software can insure that Privad is operating according to design. This shifts trust from the Privad client to the third-party software, which could be for instance the user's anti-virus software which, by definition, is already trusted by the user.

There of course may be multiple competing brokers each with a client on a given user's computer (Figure 3). These clients could operate independently of each other, for instance with each client fully imple-

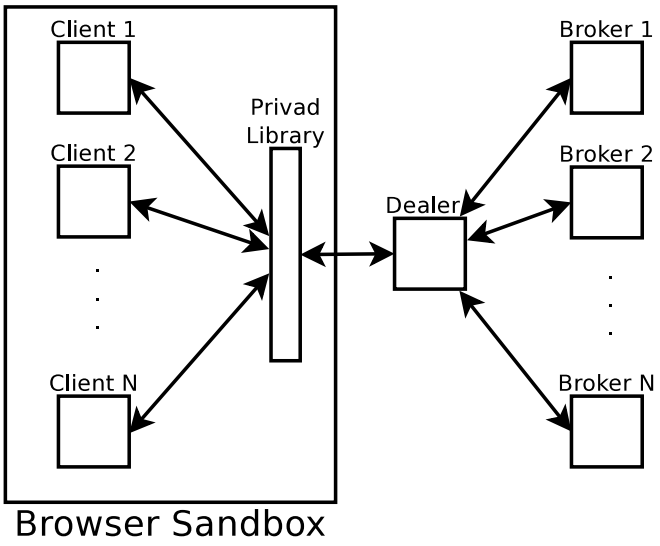


Figure 4: Multiple clients with common support

menting the Privad protocols, parsing every web-page seen by the user, and even arranging for separate dealers. Alternatively, there could be some common support infrastructure in the user's computer to handle multiple clients more efficiently (Figure 4). Clients could share a common Privad protocol implementation, common web-page parsing software. The brokers could also share a common dealer.

4 Scaling Privad: A Measurement Study

This section describes a small measurement study that justifies the design for the three Privad mechanisms, dissemination, reporting, and click-fraud. It also provides some justification for the "adware" approach to deployment.

4.1 Trace Data

We gathered two sets of traces to guide our design decisions: Google search ads, and CoDeeN click-stream.

Google data: We sampled Google search ads for a month-long period. We selected 1300 words uniformly at random from a dictionary consisting ∼100K words built from a webpage catalog [31]. We then issued a Google search query for each word roughly every 30 minutes and recorded the ads served. While we took a number of measures to discover as many ads as possible (e.g. issuing queries from a geodiverse set of planetlab nodes), we cannot determine what fraction of all ads for the chosen set of queries we managed to discover. Nevertheless, we believe the data captures the qualitative characteristics of ads (in the US), and serves only as a rough guide in quantitative matters.

CoDeeN data: We collected the click-stream pertaining to existing ad networks (Google, Doubleclick, etc.) for anonymized CoDeeN users over a month-long period. We use CoDeeN's bot-detector tool [28] to restrict our analysis to the approximately 31K human users in the dataset. We noticed, however, many false positives for bots that have evolved since the tool was first developed. We retain these bots in the data to account for bot activity one might normally expect. Altogether, the data allows us to analyze how users interact with ads (views/clicks etc.). Note that the data is inherently biased as CoDeeN users are more technically inclined than average web users[1].

In our analysis below, we point out where uncertainty or bias in our datasets affects results.

[1]Using CoDeeN for instance requires manual browser configuration

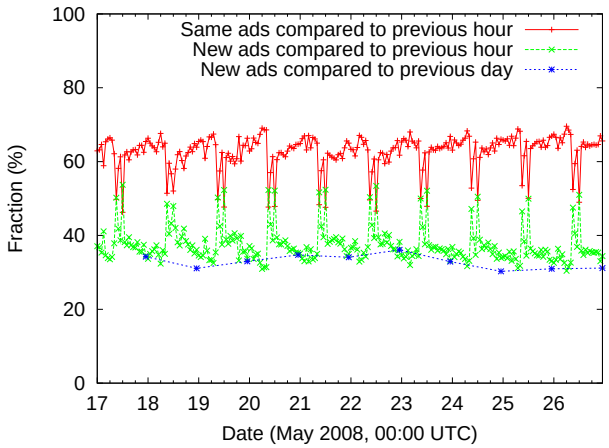


Figure 5: Ad churn in Google search ads

4.2 Ad Dissemination

Push vs. Pull: The simplest approach to disseminating ads is to flood all ads to all users. While this is ideal from a privacy perspective, we found it is not scalable. The problem is not so much the large number of ads (which could otherwise be disseminated using P2P), but rather ad churn. Figure 5 plots the fraction of ads that change from day-to-day, and hour-to-hour. Between 30%–40% ads on a given day/hour differ from those the previous day/hour. There is a daily spike at around 4am US east-coast time, when we suspect the daily budget is reset and ads that expired the previous day are reactivated. Additionally, 5%–10% of ads are permanently replaced every hour (i.e. the old ad was never again seen, and the new ad was never seen before). Thus even with optimal caching, we estimate flooding updates would require 2GB/month of compressed ad data, which is unacceptably high considering the margin for error in our estimates.

A pull based mechanism is more attractive. Ad churn does not increase client load, and additionally, turnaround for new ads is quick since the broker directly controls each ad sent. The question then is how Privad scalability compares to existing ad networks, which also use a pull model.

Request-Response vs. Pub-Sub: Existing ad networks use a request-response protocol where ads must be disseminated in real-time. This places a hard limit (few hundred milliseconds) in which to find candidate ads that match the user's profile, select the best ads, and return the results.

In contrast, a Pub-Sub approach does not require such fast responses. The broker can send out matching ads well in advance of the user visiting a publisher since the client can store the ad until needed. Further, Pub-Sub requires half the number of messages compared to request-response since the subscribe message is sent only once.

Personalization in the cloud vs. at the edge: Existing ad networks personalize ads to individual users based, for instance, on past browsing history, search history, and user preferences (some networks). Since each user is different, processing and memory requirements grow with the number of users.

Pub-Sub channels tradeoff network for less memory and processing at the broker. Since the fine-grained personalization is performed at the client, the broker only needs to perform very coarse-grained filtering — i.e. classifying ads into channel categories — to reduce network overhead. Thus processing and memory requirement are driven primarily by the (fixed) number of channels and is largely independent of the number of users (modulo maintaining channel membership, which amounts to a few bytes per online user).

The tradeoff is added network traffic since more ads must be sent as compared to existing networks, although with fairly fine-grained channels, this can be reduced to a degree. That being said, the use of crypto for the privacy aspects amplifies the cost of sending extra ads.

Cryptography Overhead:

In our Google dataset we observe the number of impressions for ads is highly skewed — a small fraction of

	Users	Ad Views	CTR	3rd-Party Toolbars	Ad Blockers
China	7308	39K	0.5 %	22 %	12 %
Saudi Arabia	6710	56K	2.7 %	40 %	9 %
United States	1420	19K	0.9 %	13 %	17 %
U.A.E	1322	8K	1.7 %	35 %	8 %
Germany	956	5K	1.5 %	7 %	19 %
Worldwide	30987	189K	2.5 %	21 %	12 %

Table 1: Online ads viewed (for top-5 ad networks), click-through rate (CTR), and the use of third-party toolbars and ad blockers by CoDeeN users.

ads (10%) garner a disproportionate fraction of impressions (80%). Second, these ads are correlated with the more generic queries in our sample set (e.g. Kitchen). Third, these generic queries have higher than average advertiser competition, which we assume results in higher costs per click. This suggests high advertising budgets and broad targeting are a likely cause, and as such are a characteristic of online advertising that can be leveraged.

Specifically, a small number of broadly targeted ads is particularly well suited to the push model discussed earlier. Supplementing the Pub-Sub mechanism with a P2P-based push (e.g. Gossip) has the potential of reducing crypto costs. Ads would be classified as *persistent* or *ephemeral*, where persistent ads are those with a broad reach and high daily budgets. The broker would use the Pub-Sub mechanism to seed persistent ads to a small number of clients that would then gossip it onward; this would significantly reduce the number of crypto operations at the broker without adding much overhead to clients due to the small number of such ads. Ephemeral ads would continue to be distributed using only the Pub-Sub mechanism. Of course, with P2P comes additional privacy concerns and attack vectors that we must consider.

Overall, we believe building a scalable privacy preserving ad dissemination mechanism is feasible. Scalability at par with existing ad networks is likely to be achievable given the benefits of Pub-Sub over Request-Response, and personalization at the edge rather than in the cloud, despite the symmetric key cryptography overhead. In addition, the possibility of reducing crypto costs further by enlisting client help looks promising.

4.3 Deploying Privad

There are two main challenges in deploying Privad: getting users to install it, and deploying the dealer function. The key to both is that Privad is verifiably private.

The user deployment for Privad is, admittedly, adware. The historical failure of adware is not due to its showing ads per se: Google and others have proven that ad-based businesses are viable. Rather, arguably the problems of adware were two-fold. First, it degraded user experience, either by showing ads in a disruptive way, or by consuming excessive system resources. Second, much adware was simply sleazy: installed without user knowledge, extremely difficult to uninstall, or outright spied on user data. This rightfully led to anti-virus companies identifying and destroying adware.

The key to Privad deployment is avoiding all the pitfalls of adware. Privad ads can be placed where they already exist—in banner ad boxes on web sites. The user won't notice any difference in the browsing experience. The fact that relatively few users install ad blockers supports the supposition that most users are ok with existing ads. Indeed, among CoDeeN users, who are presumably more technically savvy on average, fewer than 12% users use ad blockers (Table 1). Of course, the client must be implemented in such a way that it doesn't consume noticeable CPU, memory, or bandwidth, but our scaling features should make this very possible. If privacy advocates can be convinced that Privad is not only not bad, but in fact a good alternative to privacy-compromising cloud-based advertising, then there is a real chance that the anti-virus companies can be won over.

Assuming that the historic pitfalls of adware can be overcome, getting users to install it is relatively straightforward. For instance, a surprising (to us) number of users are willing to install plugins that offer minimal value to the user, for instance a toolbar or performance enhancer. In the CoDeen dataset, 21% of

users installed one or more third-party toolbars (e.g. Alexa). For non-technical users this number could well be higher.

Bundling the Privad client with various freeware applications ranging from the low-end (screen-savers) to the high-end (Firefox browser) can bring in more users, albeit at some per-user expense. Ultimately, however, bundling with computer manufactures or operating system companies will yield the most users. Again, the key to this is winning over privacy advocates and possibly even regulatory agencies.

Unlike the broker, the dealer is not motivated by profit. We envision that the dealer would be funded through a special tax levy imposed on the broker, and operated either by privacy advocacy groups, government agencies, or some combination of both (i.e. operated by one with oversight by the other). Again, the key is in convincing privacy advocacy groups of the value of Privad relative to the status quo.

5 Privad Details

5.1 Attributes

Attributes are a general mechanism that is used to implement demographics, interests, and keywords. There are two design choices: flat or hierarchical. Under a flat scheme, there is no relationship between attributes; so, for instance, Tennis and Sports would be unrelated and a tennis ad would have to explicitly list sports to be shown to people interested in all sports. In contrast, hierarchical attributes are more concise. An ad targeted to `sports.tennis` would be shown both to people interested only in tennis as well as those interested in all sports. In Privad, we opt for hierarchical attributes.

An ad targeting attribute A_a matches a profile with attribute A_p if A_p is contained in the attribute hierarchy rooted at A_a. Note matching is asymmetric. An ad for `sports.tennis` matches a user interested in `sports`, but not vice-versa.

A second design decision is whether attributes are pre-defined or not. A pre-defined attribute hierarchy (which can change over time) reduces ambiguity. For instance, the broker can determine that Java is an ambiguous term and can define three separate attributes for Java the programming language, the island, and the coffee. The downside is that creating an exhaustive attribute hierarchy is a non-trivial task. In Privad, we take a hybrid approach: the broker defines much of the attribute hierarchy along with a special node (`dyn`) under which attributes can be created dynamically. The primary use of the `dyn` hierarchy is to support keywords.

Keywords are mapped onto attributes as follows. Unambiguous keywords and keywords that can be disambiguated based on context are mapped to a pre-defined attribute if one exists. Otherwise, they are mapped to an attribute in the `dyn` hierarchy. Thus the keyword Ping Pong is mapped to `sports.tabletennis`, while the keyword Bob's Autos is mapped to `dyn.Bob's Autos`.

5.2 User Profiling

A user's profile is a set of attributes that describe the user. As mentioned, the profile is constructed by the client software by monitoring user activity. There are three basic approaches to profiling: crawling, scraping, and metadata. We discuss each below.

Crawling: The simplest approach to profiling users is for the broker to crawl the web and pre-classify websites. This is closest to the approach taken today. The client would (anonymously) query the attributes associated with a webpage visited by the user. One advantage of crawling is being able to use complex algorithms at the broker to associate attributes to arbitrary content. The disadvantage, however, is that it doesn't work for webpages that require the user to log in. Or for desktop applications used by the user.

Scraping: The client software scrapes information from webpages visited by the user and from desktop applications. This is easily done for websites (and applications) that present structured information that maps directly to a pre-defined attribute. Examples include online social networking, shopping, and travel sites, the user's local audio and video library, etc. We envision the client will have a modular architecture with website and application specific plugins. The plugins will be written and kept up-to-date by the broker.

Figure 6: Message exchange for Pub-Sub ad dissemination. $E_x(M)$ represents the encryption of message M under key x. B is the public key of the broker. C is a symmetric key generated by the client for only this subscription.

In contrast to crawling, scraping works with websites that require the user to log in. However, mapping less structured content (e.g. blogs, search terms, word documents) to pre-defined attributes on the client is hard because of practical limits on the complexity of the client. One can imagine a dictionary or a small natural-language model mapping text fragments to attributes being feasible in the client, however, a more comprehensive model would require outside assistance.

Metadata: Finally, websites can directly embed profile attributes as metadata in the webpage, which the Privad client can use directly. Local applications can directly communicate profile attributes to the client. The broker would incentivise this by offering a portion of the ad revenue to the website or application providing profile information (separate from publisher that provided ad space). To this end, the client would keep track of which sources contributed profile information that ultimately led to a click, and report it as part of the anonymous reporting mechanism. By rewarding better profiling, Privad would, somewhat paradoxically, lead to *fewer* ads and a better user experience on websites with highly targeted content.

User/Social feedback: While Privad does not rely on user feedback for profiling, nonetheless, the client can make use of it when available. One can imagine user-interfaces as simple as a plus/minus icon near an ad that reinforces or weakens the profile information that led to that ad being selected. User feedback could additionally influence profiles of the user's friends.

5.3 Ad Dissemination

The most privacy-preserving way to disseminate ads would be for the broker to transmit all ads to all clients. In this way, the broker would learn nothing about the clients. In a previous study where we measured Google search ads [16], we concluded that there are too many ads with too much ad churn for this kind of broadcast to be practical. In the study we observed that the number of impressions for ads is highly skewed: a small fraction of ads (10%) garner a disproportionate fraction of impressions (80%). Furthermore, this 10% of ads tend to be more broadly targeted and therefore of interest to many users. It may therefore be cost effective to disseminate only this small fraction of ads to all users, for instance using a P2P mechanism like BitTorrent. For the remaining 90% of ads, however, a different approach is needed. Therefore, we design a privacy-preserving Pub-Sub mechanism between the broker and client to disseminate ads.

The Pub-Sub channels are defined by a nested interest category and limited broad demographics such as geographic region, gender, and language (e.g. `sports.tennis.rackets.Wilson + location.us.ny.-ithaca + gender.male + language.en`). The interest and demographics are chosen in such a way that each channel has a large number of subscriptions for privacy, and yet acceptable scalability is achieved.

The Pub-Sub exchange consists of a request to join a channel, followed by a stream of ads being served to the client (Figure 6). The request is encrypted with the broker's public key (B) and transmitted to the dealer. The request contains the Pub-Sub channel (*chan*), and a symmetric key C generated by the client and used by the broker to encrypt the stream of ads sent to the client. The dealer generates a request ID (*Rid*) unique to the subscription request, stores a mapping between *Rid* and the client IP address, and appends the *Rid* to the message forwarded to the broker. The broker attaches the *Rid* with ads published, which the dealer forwards to the associated client. The broker may set ads to expire after a few days to ensure freshness.

Each subscription has a different symmetric key (C), thus preventing the broker from associating multiple subscriptions from the same user and therefore building up a profile of the user. Additionally, the client staggers bursts of subscriptions by adding random delays (e.g. when the client starts up) to prevent the broker from correlating them in time and associating them with the same client.

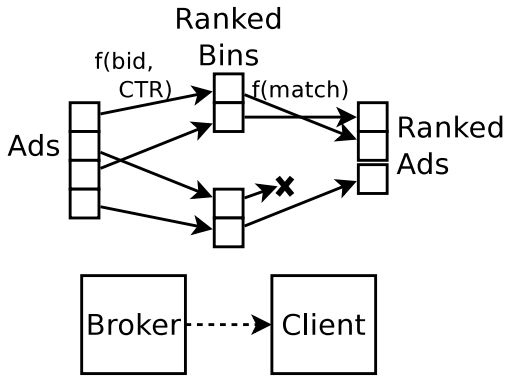


Figure 7: Design-I: Simple Auctions. For each Pub-Sub channel, broker bins ads by bid and global click-through rate. For each bin, client ranks ads by quality of match, filtering ads that don't match the user.

The broker determines which ads should be sent. For instance, the broker stops sending ads for an advertiser when the advertiser nears his budget limit. Note that not all ads transmitted are appropriate for the user, and so may not be displayed to the user. For instance, an ad may be targeted towards a married person, while the user is single. Because the subscription does not specify marital status, the broker sends all ads independent of marital status or other targeting, and the client filters out those that do not match. Over time, the broker can estimate the number of ads that must be sent out for a particular advertiser to generate a target number of views and clicks.

5.4 Ad Auctions

Auctions determine both which ads are shown to the user and in what order. The ultimate goal of any auction mechanism is to maximize expected revenue for the broker, which depends both on advertiser bid and likelihood the ad will be clicked. The challenge, of course, is in doing so while preserving privacy of the user, as well as the advertiser's bid. Below we present two auctions designs that meet our requirements. The first design implements a basic auction. The second design, which is more complex, implements precisely the GSP auction [10] used by Google today within the confines of the Privad model.

5.4.1 Design-I: Simple Auctions

The simplest approach is for the broker and client to conduct auctions at ad dissemination time (Figure 7). For each Pub-Sub channel, the broker bins ads by bids and sorts the bins in decreasing order. Top ranked ads are sent to clients subscribed to that channel. The client sorts ads within each bin based on the quality of the match. When an ad box is encountered, the client picks a channel to show ads from; the ads are shown in ranked order. The advertiser is charged the amount he bid for each click report (and a fraction of the bid for each view report). The broker periodically repeats this process, excluding advertisers that reach their budget limit. Note the ranking function may be more complex. For instance, the broker may take into account the aggregate click-through-rate (CTR), for instance, putting an ad that bids half as much as another ad but is thrice more likely to be clicked in a higher ranked bin than the other ad.

While this approach is extremely simple and doesn't require any changes to the protocol described thus far, the simple auction is coarse-grained. First, ads in different channels are not compared even if the client subscribes to multiple channels. Second, per-user information is used only to rank ads within one bin and not across bins. And third, the auction is volatile; this is inherent with first price auctions (where a bidder pays exactly what he bid) in a setting where bids can be updated and the outcome tested quickly. To illustrate: consider advertiser A bids \$2 and is ranked first, while advertiser B bids \$1 and is ranked second. From A's perspective, if he lowers his bid to \$1.01, he pays 99¢ less without any change in the auction outcome. A can

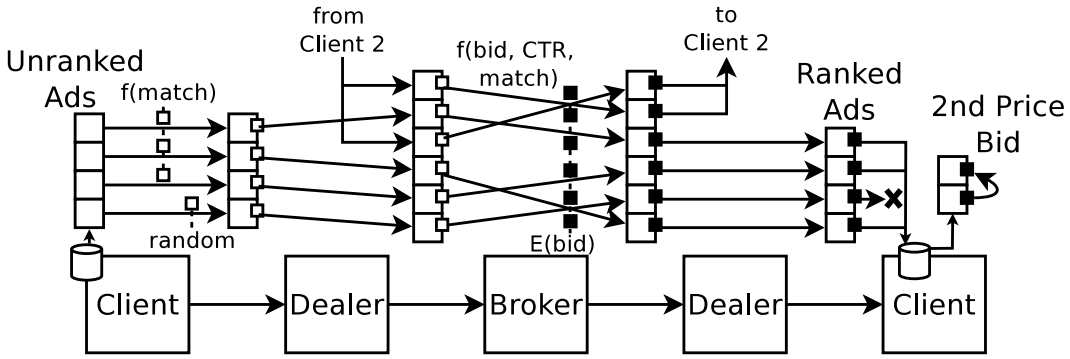


Figure 8: Design-II: Combined Auctions. Client annotates ads (across all channels) with quality of match, or random number if the ad doesn't match the user. Dealer mixes annotations from multiple clients. Broker ranks ads by bid, global click-through rate, and match quality, and annotates the result with opaque bid information. Dealer slices auction result by client. Client filters out non-matching ads. Client reports second-price bid on click.

determine his most optimal bid by trial and error. At which point, B can determine by trial and error that by bidding only 2¢ higher, B is ranked first. This constant trial and error driven by real financial incentives results in volatile prices and a constantly changing ranking of ads, which interacts poorly with our goal of caching the auction result at the client.

5.4.2 Design-II: Combined Auctions

In the combined approach (Figure 8), the broker conducts the auction in a separate exchange. First, ads are sent to clients as originally described. The broker attaches a unique instance ID (Iid) to each copy of the ad (not shown in figure). For each ad, the client computes a coarse score, typically between 1 and 5, as follows: for ads that match the user, the score reflects the quality of match with 5 signifying the best possible match. For ads that don't match the user, the score is a random number. To rank ads, the client sends $(Iid, score)$ tuples for all ads in the client's database to the dealer. The dealer aggregates and mixes tuples for different clients before forwarding them to the broker. The broker ranks all the ads in the message. The ranking is based on bids, CTR, and client score. Note this ranking contains all ads from the same client in the correct order, interwoven with ads for other clients (also in their correct order), but doesn't allow the broker to learn which ads matched a given user, or which channels a given client is subscribed to. The broker returns this ranked list to the dealer. The dealer uses the Iid to slice the list by client and forwards it to them. The client discards the ads that do not match the user, and stores the rest in ranked order. Note that the entire exchange is unencrypted; since Iids are single-use, they do not leak ad information to the dealer.

The issue of volatility, is solved using second-price auctions [10]. In second price auctions, each bidder is charged the next highest bid. Thus the highest bidder pays the second-highest bid, second-highest bidder pays third-highest bid, and so on until the lowest bidder that pays some minimum bid (typically 1¢). The second-price outcome is identical to the steady state behavior of the first-price auction without the associated volatility. However, a straightforward application of second-price auctions at the broker does not work because the broker does not know which ads are from the same client, much less which ads will be discarded as they do not match the user.

Second-Price Auctions. To perform second-price auctions, the broker encrypts the bid information with a key known only to the broker and sends it along with the ad. When a set of ads are chosen to be shown to the user, the client copies the encrypted bid information from ad $n+1$ to ad n. This encrypted bid information is sent as part of the click report, which the broker decrypts to determine what the advertiser should be charged. Second-price bid information is not sent for view reports for privacy reasons.

Overall, the combined auction duplicates *precisely* the GSP auction used by Google [10] without sacrificing client or bid privacy.

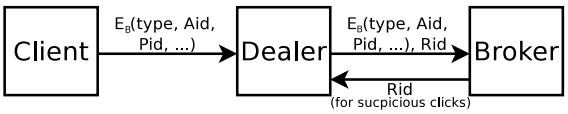


Figure 9: Message exchange for view/click reporting and blocking click-fraud. B is the public key of the broker. *Aid* identifies advertiser. *Pid* identifies publisher website or application on which the ad was shown. For second-price auctions, the opaque auction result is included. *Rid* uniquely identifies report at dealer.

5.5 View/Click Reporting

Ad views and clicks, as well as other ad-initiated user activity (purchase, registration, etc.) needs to be reported to the broker. Each report contains the type of event (view, click, etc.), as well as the Ad ID (*Aid*) and Publisher ID (*Pid*). The *Aid* uniquely identifies the ad, and the *Pid* identifies the website or webpage on which the ad was displayed. If the ad was shown through some other means (within the GUI of an application, or as ad placement within a virtual reality game), the *Pid* identifies this.

As with the subscription message, the report is encrypted with a public key belonging to the broker and transmitted to the dealer (Figure 9). To assist the broker in defending against click-fraud, the dealer attaches a unique report ID (*Rid*) to the message, and briefly stores the mapping of *Rid* to client IP address. As before, if the client has multiple reports to send at once, for instance because multiple ads appeared on the same web page, the client staggers them by adding random delays to prevent the broker from correlating them.

5.6 Anonymizing the Click

If when a user clicks on an ad, the user goes directly to the advertiser's website, then the advertiser can know a significant amount of information about the user: some demographic information (whatever matched the ad) and the user's IP address. Worse, if the advertiser can link multiple clicks, for instance because they came from the same IP address or supplied the same cookie, then the advertiser can build up a profile of the user derived from the Privad profile.

One approach to mitigating this would be for the client to go through a standard web proxy. The problem here is that the web proxy is in a position to know a great deal about the user. Of course, this is true in any event for web proxies today, and that doesn't stop many people from using them, but Privad makes the situation slightly worse by making it possible for the web proxy to glean the user's profile.

A stronger approach is to use the dealer/broker infrastructure itself to anonymize clicks. A simple method is for the click report to contain a symmetric key that allows the broker to establish a secure transport connection with the client (via the dealer). This connection can be used by the broker to proxy the click and (possibly) subsequent browsing by the user. To avoid the overhead and delay of public key encryption at click-time, the client could pre-establish the symmetric key (Figure 10). This is done by having the client transmit a (public-key encrypted) record containing a symmetric key and a $KeyID$. The record does not have to be decrypted in real time, allowing for instance the public key encryption offload mechanism to be used. When the user clicks an ad, the advertiser URL and $KeyID$ is transmitted to the broker, encrypted with the symmetric key. The broker uses the key to establish a secure connection with the client, proxied to the URL.

An open question is, for how long does the broker proxy the client/advertiser exchange? At one extreme, the broker could proxy only the advertiser landing page, allowing the user to continue at his or her own risk afterwards. Indeed, this could involve an indication to the user as to what the advertiser might know about the user at that point. At the other extreme, the broker could proxy the entire exchange. There is a danger here, however, in that the user could give up some personally identifying information during the exchange, which would then be revealed to the broker (as well as the advertiser). This, however, could be mitigated by having the advertiser establish a secure connection with the client (via the broker/dealer).

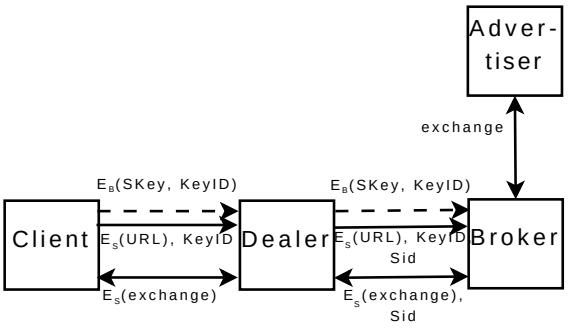


Figure 10: Proxied click (without SKey pre-caching)

5.7 Detecting Click-Fraud

There is no silver bullet for defending against click-fraud. The approach we take is *defense in depth*. We design a number of overlapping detection strategies. Each detection strategy can be fooled with some effort, but together they raise the bar.

Thresholds: The dealer tracks the number of subscriptions, and the rates of view and click reports for each client. Clients that exceed thresholds set by the broker are flagged as suspicious. The broker may set higher thresholds for ISPs that NAT multiple users, or public proxies.

An attacker can stay below the threshold by distributing the attack across several dealers, or by using a botnet to distribute the source of the attack. In case of botnets, rate-limits at least make it harder for the attacker to launch multiple simultaneous attacks.

Blacklist: Dealers may share a blacklist of blocked clients. This prevents a client blocked at one dealer from using a different dealer to continue the attack. Additionally, dealers flag clients appearing in public botnet/infection databases maintained by third-parties such as anti-virus vendors, and network telescope operators.

This limits the length of time the attacker can use a compromised host to launch click-fraud.

Honeyfarms: The broker operates honeyfarms that are vulnerable to botnet infection. Once infected, the broker can directly track which publishers or advertisers are the subject of click-fraud attacks and take corrective action.

Honeyfarms allow probabilistic detection of attacks that manage to avoid other approaches. Attackers may of course attempt to detect honeyfarms, but that can be made arbitrarily hard by the honeyfarm operator.

Historical Statistics: The broker maintains a number of per-publisher and per-advertiser statistics including volume of view reports, and click-through rates. Any sudden changes in these statistics cause clients generating the reports to be flagged as suspicious. However, the broker does not know which client generated the report. Thus to flag the client, the broker sends the *Rid* associated with the report back to the dealer, which uses the stored mapping to identify the client IP address.

Historical statistics forces the attacker to gradually build up the attack. This buys time for the other detection mechanisms to kick in.

Bait Ads: The broker occasionally disseminates "bait ads", which can loosely be described as captchas for advertising. Bait ads contain the targeting information of one ad, but the text of a different ad. For instance, a bait ad may advertise a tennis racket, but be targeted at opera enthusiasts. The broker expects a very small (but non-zero) number of such ads to be clicked by humans. A bot, however, would trigger multiple bait ads and be flagged as suspicious.

Bait ads can be designed to expose attacks benefiting specific publishers, or a group of similar publishers. Additionally, bait ads can be generated on demand, and in large numbers (quadratic in the number of ads). Note, bait ads take up real ad spaces on web pages but are not expected to generate many clicks, and as such have a negative impact on revenue. However, their volume and frequency can be controlled by the broker to

Crypto Operations

ENCRYPT(MSG, PUBKEY)
: Encrypts MSG with public-key PUBKEY.

GENSYMMETRICKEY()
: Generates a new symmetric key.

GENASYMMETRICKEYPAIR()
: Generates a public-private keypair.

Network Operations

SEND(MSG, DEALER)
: Sends message MSG to dealer DEALER.

ONRECV(HANDLER)
: Registers a handler for incoming messages.

Table 2: Reference monitor API

bound their impact on revenue.

An attacker can identify bait ads if it can detect the mismatch between the ad text and the ad targeting information; this is not necessarily hard in a theoretic sense, but is in practice non-trivial. Detecting bait for image ads is harder still. The attacker could use cheap human labor to detect bait as attackers do today for captchas, but the fact that bait ads are based on the *meaning and context* of words, which requires English comprehension to detect, rather than distorted script, which requires only pattern matching, may present a very different value proposition in non English-speaking countries.

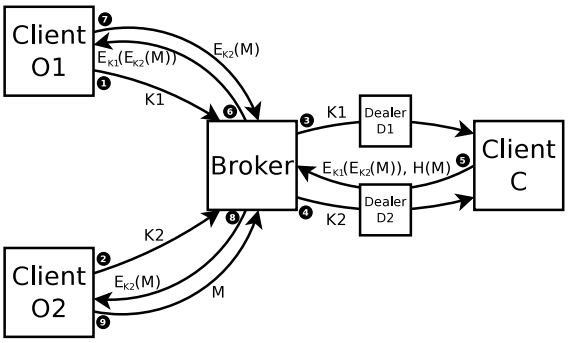
5.7.1 False Positives and False Negatives

False negatives, i.e. not identifying an attacking client, are more important than false positives, i.e. suspecting a legitimate client. Note the broker does not directly lose money to click-fraud since the advertiser pays for each click. Rather more importantly, click-fraud impact's the broker's reputation making it harder to retain advertisers and attract new advertisers in the long run. On the other hand, the advertiser gains from false positives since the user is still directed to the advertiser's website, but the advertiser is not charged for the click. Undercharging the advertiser, while undesirable, is much less severe than losing reputation. By overlapping bait ads, historical statistics, honeyfarms, blacklists, thresholds, and yet more sophisticated mechanisms that one might imagine, the broker progressively reduces false negatives.

To reduce false positives, the dealer waits until a client has been flagged several times before silently dropping reports from that client. When a client crosses the threshold, past reports may be invalidated retroactively since the dealer keeps a record of recent *Rid*s associated with the client. Thus reducing false positives does not come at the cost of false negatives. Lastly, to account for compromised clients being disinfected, the dealer unblocks clients after some time. If the client's click-fraud behavior persists, the length of time the client is blocked is increased.

5.8 Reference Monitor

Table 2 lists the complete API exposed by the reference monitor. The client uses the monitor's ENCRYPT function to encrypt all messages. Additionally, the client uses the monitor's GENSYMMETRICKEY functions to generate symmetric keys for subscribe messages. The reference monitor validates message contents to ensure the client does not leak sensitive profile attributes. Additionally, the monitor ensures that no two subscribe messages contain the same key, and that the keys were generated using the functions provided. By having the monitor generate keys and perform the encryption we reduce the possibility of the client passing information covertly to the broker (e.g. through random bits in generated keys, or through randomized padding in the encrypted message).



#	From	To	Message
1	Client $O1$	Broker	$K1$
2	Client $O2$	Broker	$K2$
3	Broker	Client C (thru $D1$)	$K1$
4	Broker	Client C (thru $D2$)	$K2$
5	Client C	Broker (thru $D2$)	$E_{K1}(E_{K2}(M)), H(M)$
6	Broker	Client $O1$	$E_{K1}(E_{K2}(M))$
7	Client $O1$	Broker	$E_{K2}(M)$
8	Broker	Client $O2$	$E_{K2}(M)$
9	Client $O2$	Broker	M

Figure 11: Offloading public-key operations to idle clients. $K1$ and $K2$ are public-keys created by clients $O1$ and $O2$ respectively. $E_y(x)$ represents the encryption of x under key y, and $H(x)$ represents a cryptographic hash of x. At the end of the exchange, broker learns M without performing any public-key operations; clients $O1$, $O2$ and broker do not learn identity of client C; dealers $D1$, $D2$ do not learn M.

The browser sandbox allows only the reference monitor to perform network I/O. The client uses the monitor's SEND function to send messages. The monitor ensures the recipient is an authorized dealer, and that any encrypted information is a result of a recent call to ENCRYPT. Finally, the monitor is allowed to arbitrarily delay messages or add jitter to further reduce the possibility of the client covertly sending information by using timing as a covert channel. For this reason the Privad protocol is designed to be delay tolerant — all operations are asynchronous, and no message requires an immediate response.

The reference monitor API is designed to be extremely small and simple so that correctness can be verified manually. We envision reference monitors will be open-source — created by privacy-advocates, anti-virus vendors, or browser vendors, and be verified by one of the others.

5.9 Offloading Public-Key Operations

We present an optimization to Privad that reduces broker overhead by leveraging idle clients to perform pubic-key operations. Figure 11 illustrates this *offload* protocol. Idle clients ($O1$ and $O2$) generate a public-private key pair. The public key is sent to the broker. The broker mixes these keys. A client C that wishes to send a message (M) to the broker first requests two offload keys through different dealers. These keys are for this message only; the client must request additional keys for another message. The client encrypts M with the two offload keys: thus if the keys are $K1$ and $K2$ the encrypted message is $E_{K1}(E_{K2}(M))$. The client sends the encrypted message and a hash of the original message ($H(M)$) through the dealer to the broker. The broker sends the encrypted message to the client that holds the private key for $K1$; the client responds with the decrypted result ($E_{K2}(M)$). The broker then sends the message to the client that holds the private key for $K2$. The client responds with the original plain-text M. The broker validates the decryption by computing the hash of M and comparing it to the value sent by the client.

At the completion of the above exchange, the broker learns M without having performed any public-key operations. Instead, the broker needs only verify the hash, which is significantly faster. As before, the dealer

protects C's identity from the broker (as well as from $O2$). Note $O2$ cannot target a particular client since the broker mixes keys. Indeed, $O2$ cannot learn anything the (untrusted) broker cannot learn thus preserving the privacy properties of the Privad protocol without offload. Note also that even though the keys are sent in the clear through the dealers, a single dealer cannot mount a man-in-the-middle (MITM) attack since the message is encrypted with two keys. The typical solution to MITM involving certificates doesn't apply since malicious dealers could arrange to get their keys certified by masquerading as a legitimate client. Finally, note that the broker still cannot link different messages from the same client since the client requests a new pair of offload keys for each message, and the broker cannot link multiple requests for offload keys from the same client. For a thorough privacy analysis of the offload protocol see [15].

6 Privacy Analysis

6.1 A brief summary

Of the system players (user/client, broker, dealer, publisher, and advertiser), we are primarily interested in user privacy. There are also advertiser privacy concerns (advertiser targeting, bidding, and budget information).

Broadly speaking, the user interacts with publishers, the broker, and advertisers. Compared to today's advertising systems, Privad changes nothing about how the user interacts with the publisher. Any advertising system with fine-grained targeting, including Privad, can however effect how the user interacts with the advertiser. In particular, when the user clicks on a finely-targeted ad, the advertiser potentially knows a great deal about the user. Even if the user uses a web proxy, thus hiding its IP address, the knowledge provided by targeting can be exploited in interactions with the advertiser. This can be mitigated somewhat by limiting the granularity of targeting. Ultimately, however, it should be possible for the broker to provide anonymity between the user and advertiser, as described later in this section.

Privad provides the user with anonymity: no information learned about a user can be associated with the user's personal identity, either through internal or external means. While this isn't the only possible definition of privacy, nor the strongest [6], we believe that it is a level of privacy that would satisfy most users. More to the point, we hope and expect that it will satisfy privacy advocacy groups, and we are currently in the process of discussing Privad with some of them.

Privad provides user anonymity through three basic techniques. First, no Personal Identifying Information (PII) other than IP address is explicitly leaked by the client. This is validated by the reference monitor. Second, the dealer, which knows the IP address, has no access to any user information. Likewise the broker, who has access to user information, does not know the IP address. Finally, the individual bits of user information that are provided to the broker cannot be linked together. This prevents the broker from compiling a complete user profile, which effectively distinguishes a single user, and then using external means to personally identify the user.

Privad does not protect against some threats. While it does not require trust in either the dealer or the broker, it does not protect against collusion between the broker and the dealer. If this is considered too great a threat, then additional dealers can be added in such a way that collusion is required between all dealers and the broker. Privad does not protect against malware running on the client. While Privad makes it somewhat easier for malware to gather user information (i.e. by looking at the profile), it does not fundamentally alter the ability of malware to gather private information. Privad does require trust in the reference monitor, which sees all information that passes between the client and the dealer/broker. The role of the monitor, however, is limited, and its operation is simple. Therefore, it is expected that the security properties of the broker can be verified by hand.

In the remainder of this section, we use the terms principal and recipient. The *principal* is the the person or organization that has a piece of information (a datum) to be kept private. The *recipient* is the person or organization that may learn the datum.

6.2 Advertiser privacy

While we are primarily concerned with user privacy, we also need to consider advertiser privacy. Advertisers would like to keep details about their advertising campaigns private. These include ad targetting information (interest categories, keywords, or demographics), the amount it bids for ads, as well as its overall advertising budget.

With current advertising systems it is possible to learn at least some of an advertiser's targetting information. To do so, the recipient would have to make a hypothesis as to what keywords the advertiser is targetting, then try some tests to see if the hypothesis is correct. For instance, the recipient could search for those keywords and see if the advertiser's ads appear. The recipient could also see if the ads show up on webpages that are found by searching on those keywords, though in this case the recipient can't be sure that it is those keywords that caused the ad to appear. With Privad, the process of learning an advertiser's targetting information is similar, though easier. The recipient would make a hypothesis as to what interest categories are being targetted. In many though not allcases, this would be quite obvious because interest categories are aligned with products and services. The recipient then joins the appropriate interest channels. The ads received for the advertiser will have the targetted demographics attached.

With current advertising systems, it is also possible, though costly, to learn how much an advertiser bids for certain keywords. This is done by competing with the advertiser for those keywords, and seeing what price beats the advertiser. Privad does not change this.

It is hard to determine the overall budget an advertiser has with current systems, and Privad does not change this. The remainder of this section focuses on user privacy.

User opt-in: Before continuing, it is important to point out that ultimately a user can protect his or her privacy vis-a-vis Privad by simply not opting in. We take it as a given that the user derives some benefit from installing Privad, for instance be being able to run an application that Privad is bundled with, or by getting shopping discounts, low-cost ISP service, or some other benefit. In what follows, we assume that the user has opted in in exchange for some unspecified *benefit*.

6.3 User privacy

User privacy, ultimately, can only be defined by each user, on a per datum per recipient basis. In other words, for any given datum, a user may wish to keep that datum secret from specific recipients. The user of course may be happy to reveal other information to that receipient, or may be happy to reveal that datum to some other recipients. We are primarily concerned with the impact of information that is created or compiled by Privad, i.e. the user profile.

Malware: One impact is that any software already running on the client machine (i.e. malware) may learn information more easily than it might otherwise. Whereas in the absense of the user profile, the malware would have to search for certain types of files, monitor the user's web activity, and so on, a Privad client will have already done some of this work. This is one reason that it is important that clients obstain from gathering information that would 1) be regarded as sensitive by a majority of users, and 2) might be of particular value to others. Medical information or information about what financial institutions a user uses are examples. Ultimately, however, Privad doesn't impact what information can be gathered by malware, and so preventing information leakage to malware is not a primary concern of Privad.

As an aside, it is worth noting that by definition the reference monitor is software that is running on the user's computer that could potentially gather and leak private information. While this suggests trust in the reference monitor, in fact the role of the reference monitor is quite limited, to checking the contents of the various Privad messages and encrypting or decrypting them. In addition, there are no trade secrets that the reference monitor needs to protect, so its code can be open source. As a result, it is reasonable to expect that the reference monitor code could be hand-checked for correctness.

Information leakage to an advertiser: Another concern is the impact of a fine-grained user profile on information leakage to an advertiser. By way of example, suppose that two of the user attributes collected by the client are 1) whether the user has any given disease (say AIDS), and 2) whether the user has health insurance. By targetting ads to these attributes, companies could exploit this information in a number of

ways, for instance to avoid selling insurance to people that appear to have pre-existing conditions (even when the customers try to hide the existence of those conditions, or in fact don't have the conditions), or to prey on desparate people with certain diseases and no health insurance.

Note that this form of privacy loss applies to any targetted advertising system, not just Privad. Even if an advertising system is completely private in the sense that it doesn't leak any information to components of the advertising system per se, once a user clicks on a well-targetted ad, the advertiser knows something about that user. Though strictly speaking outside the scope of Privad, one thing a user can do to mitigate the privacy loss resulting from clicking on a well-targetted ad is to use a web-proxy. This, combined with cookie deletion, can increase anonymity of the user from the advertiser. Never-the-less, even if the user has this kind of anonymity, once the user clicks the advertiser knows information about the user that the user potentially considers private, and can exploit this information in the ensuing dealings with the user.

Note that it isn't purely the profile category (interest or demographic) alone that erodes privacy. Rather, it is the combination of categories and advertiser. For instance, regarding the example of AIDS and health insurance, it seems to us appropriate for a company that sells AIDS medicine to be able to target people whose profile shows an interest in AIDS. It does not, however, seem appropriate for a health insurance company to target people interested in AIDS.

Generally it is up to the broker to limit the target categories, or combinations of categories and advertisers. This can for the most part be monitored. For instance, brokers, advertisers, and privacy advocates could work together to define a set of guidelines for categories and combinations of categories and advertisers. Privacy advocates could also provide users with reference monitors that allow or disallow ads according to these guidelines, tunable by the user. Of course, taken too far, this could simply degenerate into ad blocking. As a result, there needs to be some give-and-take between the interests of advertisers and those of users so that on one hand advertising continues to be a source of revenue for web sites while at the same time minimizing loss of privacy.

A broker and advertiser could collude to fool users into thinking that a category means something that it doesn't. For instance, say an advertiser wants to target people with some disease and no health insurance, but privacy advocates wish to prevent it. The broker and advertiser could collude to agree that the category "tennis strings" really means "AIDS", and the category "lima beans" really means "no health insurance". The advertiser then targets its ads to "tennis strings" and "lima beans", and the ad gets shown to people with AIDS and no health insurance. While this is in principle detectable, the cost of doing so would be quite high. It would ultimately require humans to monitor categories and see if the ads shown make sense. By the same token, the risk for a broker and advertiser getting caught should also be relatively high (i.e. bad publicity or a government-levied fine).

Information leakage to the broker or dealer: Given that privacy leakage to advertisers is a problem common to all targetted advertising systems, we are primarily concerned with information leakage to the broker or the dealer, both of which are non-trusted entities, and both of which handle all messages in and out of clients and so are in a position to obtain lots of user information.

On the assumption that the client and reference monitor are operating correctly, let's first consider what potential exists for privacy leakage by looking at the sum total of information that leaves the client. The client transmits:

- Channel subscriptions, which contain the client interests and limited broad demographics,
- Reports, which indicate which ads the client has seen or clicked, and some of the websites the client has visited. Since the targetting information associated with ads have demographic information, user demographics can be inferred from a list of ads shown. The inferrence isn't perfect, since there is no indication of which demographic information caused the ad to match the user. Nevertheless, given enough reports, the demographics could be accurately determined.

In short, the potential for privacy leakage includes:

1. user interests,

2. user demographics,

3. some of the websites visited, including frequency, and

4. the IP address of the user.

It is worth noting that the leakage does not include any Personal Identifying Information (PII) per se (i.e. name, home address, social security number, etc.). It also does not include URLs and HTTP headers, which reveal the specific web pages visited as well as other information that can be used to derive user PII such as cookies and referer headers [23]. Therefore, assuming that Privad avoids highly-sensitive profiling, even the *potential* for leakage in Privad is less than the *actual* leakage in existing advertising and tracking systems.

We assume that a user doesn't care if a recipient sees the client information, as long as the recipient can't associate the information with the user. For example, a user interested in cancer treatment doesn't care if some recipient (the broker or dealer) knows that *someone* is interested in cancer treatments, as long as that recipient cannot discover who that someone is.

Note that this model of privacy is different than, and weaker than, that assumed by Private Information Retrieval (PIR) [6]. The goal of PIR is to prevent information servers from even knowing what information was queried. In other words, with PIR, no system would even know that *someone* is interested in cancer treatment.

Our model of privacy leads to two goals:

1. **G1:** No recipient can link any client information with client identity (i.e. IP address), and

2. **G2:** No recipient can link different pieces of information from a given client together.

The need for the first goal is self-evident. The need for the second goal is to prevent multiple pieces of information taken together from uniquely identifying the user. For instance, it has been shown that 87% of Americans can be uniquely identified from the combination of birth date, zip code, and gender. The more information Privad is able to link to a given client, the more likely it is that the client can be uniquely identified.

In the Privad design, each report represents a single click or view, and as such contains a single Ad ID and a single Publisher ID only. Recipients in Privad must not be able to link multiple reports to the same client. Likewise, in Privad each subscription is for a single interest category. Recipients in Privad must not be able to link multiple subscriptions to the same client. Of course, recipients must not be able to link reports with subscriptions.

While the dealer can link multiple messages to a client, it cannot decrypt those messages, and so has no information to link. The broker can decrypt the messages, but since the dealer hides the IP address and sends no other information that could link one message to another, the broker also cannot link reports and subscriptions. Since the client delays transmission of the messages (Section 5.3), the broker cannot link reports or subscriptions received within a small window of time, for instance when a client first starts up. If the broker and the dealer collude, then both goals fail.

The need for collusion raises the bar substantially over today's status quo. If this bar is not considered high enough, then additional dealers in serial can be added, thus requiring collusion between all dealers and the broker. Note that the subsequent dealers must delay and reorder messages. This thwarts an attack whereby the broker and dealer collude, and use the timing or ordering of message received at the broker from the last dealer to associate them with messages sent by the first dealer.

In what follows, we assume no collusion between dealer(s) and broker, and we assume that the client operates correctly (i.e. no malware on the client computer). We examine a number of attacks, and describe when Privad does and does not satisfy its two linkability goals.

A1: The broker masquerades as a dealer and hijacks the client's messages. One way of hijacking the traffic is to subvert DNS [2] or BGP [4]. The solution is to require TLS between dealer and client, and to use a trusted certificate authority. The reference monitor can insure that this is done correctly, or operate the TLS connection itself.

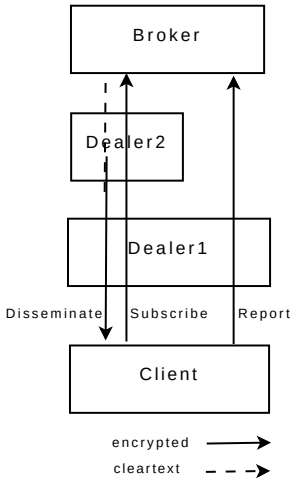


Figure 12: Client profiling defense 1

A2: In this attack, the broker creates a set of false ads each with a unique Ad ID, and feeds these ads to a single client through a single subscription. The demographics attached to the ads are designed to expose a demographic profile of the user. This can be done by representing each individual demographic type with a single ad. For instance, one of the ads can list only an age range as the single demographic. If a view report is returned for the ad, then the broker knows the age range of the user. Another ad can filter on salary range, and so on. The broker can then link the view reports by associating the Ad ID with the client. Through external means (social networks or other sources of information) the broker may be able to identify the user.

A simple defense against this is for the client to generate two subscriptions for every interest channel that it joins. It then only considers ads that it receives on both channels. This defeats the attack because it forces the broker to send the same ad on many channels in order to cause any one client from receiving copies on both of its duplicate channels. If, however, the broker sends the ad on many channels, then it has no way of correlating the reports with the same client. The problem with this defense is that it increases the number of ads distributed significantly. This is due in large part to the large number of ads from advertisers with small budgets. Such ads should only be viewed by a small number of clients, and therefore should ideally be distributed to a small number of clients. If, however, any client has to see a copy of the ad on both of its duplicate channels, the broker has to increase the number of times the ad is sent tremendously in order to insure that the right fraction of clients see duplicates.

The alternative is to establish an additional player that on one hand is able to insure that multiple copies of any ad goes out, but on the other is not in a position to benefit from this attack.

One way to do this is illustrated in Figure 12. Here, a second dealer (Dealer2) is inserted between the normal client-facing dealer and the broker. Subscriptions flow through both dealers, with both dealers keeping state so that they can disseminate ads to the correct clients. The subsciptions are conveyed to the broker encrypted as with the single-dealer design. The broker decrypts them, and then transmits the ads as well as the symmetric client key to Dealer2 in the clear. Dealer2 insures that enough copies of each ad have been sent to different clients, or holds on to any ads that don't yet have enough copies, and sends the copies once there are enough. Dealer2 encrypts the ads using the appropriate symmetric client key, and forwards them to Dealer1. When the client later sends a report, these may sent directly from Dealer1 to the broker: they do not need to traverse Dealer2.

A3: In the second-price combined auction, the broker passes the ad bid, encrypted, along with the ad. In a click report, the client returns the excrypted ad bid of the ad ranked after the clicked ad. The broker can attack linkability by encoding the Ad-ID in the encrypted field. Thus, in the click report, the broker

can link the clicked ad with the next ad shown. To the extent that the combination of the two ads reveals additional demographic information than the single ad, privacy is weakened.

We do not offer a defense against this attack, but point out that the attack itself is minor because the broker can link no more than two ads. Further, the attack is limited to clicks, which are far more rare than views.

A4: In this attack, the broker colludes with a publisher as follows. The publisher tells the broker the time that a page was served, and the IP address to which it was served. The broker then links a report that arrives shortly after with the IP address. The defense is for the client to delay reports by a random amount uniformly distributed over a relatively large time period. Depending on the rate at which pages are served, and the maximum amount of delay, the broker can at best associated a set of IP addresses with a set of reports. How large the set is depends on the page rate and delay period. The penalty for delaying a view report is that it takes longer for the advertiser to gauge the click-through-rate (CTR) of a given ad. However, the time period at which an advertiser can realistically respond to CTR is many hours at best, so a substantial delay in view reporting is reasonable.

As a variant on this attack, the publisher can shrink the set of clients by manipulating the Publisher ID *Pid* that is reported, and isolating it to one or a small number of clients. Mechanistically this could be done whether the *Pid* is a URL (minus the "searchpart" portion that comes after the "?" symbol) or a domain name. For instance, the publisher could generate unique *Pid*'s per IP address over the delay time period, thus allowing the broker to isolate the IP address given the page serve and report times. Indeed, the publisher could even covertly embed the IP address of the client in the *Pid*, thus eliminating the need to exchange timing information at all.

Fortunately, this publisher behavior is detectable. An auditor could establish a honey-farm of clients that access web pages and look for unique URLs or domain names that follow the pattern of the above attacks. Once found, the publisher administrator could be contacted to try to end the behavior. Failing this, the publisher could be published in a black-list that the reference monitor could use to block reports.

A5: In this attack, an advertiser attempts to determine a client's demographic profile so that when a client clicks on an ad, the advertiser knows the profile of the user. At worst, this could be exploited by the advertiser using external means to obtain PII on the user. Even without that, given a near-full profile, the advertiser has a distinct advantage over the user on any subsequent web transaction that takes place.

As already discussed earlier in this section, it is possible to limit the amount of demographic detail associated with any given ad. In this attack, then, the advertiser links multiple advertisements that a given user clicks on, either through IP address or cookie. Note that this attack is not specific to Privad. Any personalized advertising system has this attack.

One low-cost defense is for the client (or reference monitor) to track how many clicks have been made to the same advertiser, and to stop showing ads to the user that could end up in a click that exceeds a privacy threshold at the client. The counter to this is for the advertiser to masquerade as multiple advertisers. However, this amounts to a clear willful attempt at gaming the system (versus one advertiser just happening to get multiple clicks from the same user but not correlating them), and therefore should be easier to prosecute should the advertiser get caught. In other words, the disencentive here is punishment if caught.

A stronger and higher-cost defense is described in Section 5.6. This defense can be defeated with collusion between the advertiser and the dealer using a timing attack. The dealer can log the time when an exchange between client and broker takes place, and compare this with web access time logs at the advertiser. Because of the real-time nature of this exchange, the attack could not be mitigated using delays.

A6: In this attack, a dealer attempts to learn which websites a user is visiting as follows: The dealer establishes a botnet of clients. These clients launch a click-fraud attack on a publisher. The attack is eventually detected by the broker, when then gives the publisher a list of clients involved in the attack. Any clients on the list that are not in the botnet must be users accessing the website. There is an analogous click-fraud attack the dealer can launch to determine which clients are viewing ads by a given advertiser.

The solution is for the broker not to provide the list of suspected clients for a given attack all at once. Rather, the broker can intersperse the notifications of multiple different attacks. This way the dealer can't be sure if a client was tagged for his attack or some other attack the details of which the dealer is unaware.

Note that this can delay the time it takes to fully notify a dealer of the suspected participants in an attack.

Another defense is for the broker to add chaf: select reports at random to tag. The downside if of course that occasionally a client not involved in any attack may be tagged enough to be discounted.

7 Implementation

We have implemented the full Privad system. The system comprises a client and reference monitor implemented as a 154KB addon for the Firefox web browser, a dealer, and a broker. The client, dealer, and broker are all written in Java. We use the Google Web Toolkit (GWT) [13] to translate the Java code for the client into Javascript that is then executed by the browser. In all, our implementation consists of 8.4K lines of Java code with the client, broker, and dealer accounting for 4.3K, 800, and 300 lines respectively. The remaining code includes 2.4K lines of RPC code, interface declarations and utility methods shared across the three components.

The client implements the combined auction mechanism described in Section 5.4.2, and the offload optimization described in Section 5.9 in addition to the core Privad mechanisms (user profiling, ad dissemination, reports). The monitor is a proof-of-concept implementation that performs the encryption and network I/O, but does not validate message contents. The dealer and broker implement all the necessary mechanisms, but support only the simplest threshold-based click-fraud detection mechanism.

The monitor is written in Javascript. It uses the open-source pidCrypt Javascript library [32] for performing cryptographic operations. We use RSA with 1024-bit keys for public-key operations, and AES with 128-bit keys for symmetric-key operations; randomized padding it used to defend against dictionary attacks. For public-key encrypted messages, as is standard practice, the message is encrypted with a random symmetric-key, and the symmetric-key is encrypted with the public-key. For the offload mechanism, only the public-key decryption is offloaded; once the symmetric key is recovered, the broker finishes decrypting the message.

Platform choice. We chose to implement the client as a browser addon to enable us to scrape highly-dynamic AJAX web applications, which would otherwise be impossible from a standalone daemon or local browser proxy perspective. While browser addons are OS independent, they are, however, browser dependent. The GWT compiler simplifies the problem significantly by translating browser-independent high-level code (in Java) to browser-specific Javascript; we needed to write less than 450 lines of Firefox-specific glue code. Finally, we were concerned about Javascript performance for cryptographic operations. While browser addons can include native code, as we show later, modern Javascript engines perform sufficiently well for this to be a non-issue.

All client-dealer communication is performed over HTTP to accommodate clients behind firewalls and proxies. We use a JSON-based RPC framework (rather than XML RPC) for brevity and better performance. The broker and dealer are written as Java servlets, hosted by the Jetty webserver [30]. The broker uses standard Java cryptographic libraries for all cryptographic operations.

User profiling. The client scrapes demographic information from the user's Facebook profile, and long-term interests from the user's Google Ad preferences[2] (which are automatically populated by Google based on the user's browsing habits). These websites present structured data that is easy to scrape. We choose these two websites to illustrate how the client can integrate with existing profiling services. We are currently in the process of implementing online shopping related profiling.

Test Ads. Since we lack real advertisers to test our system with, we currently scrape ads from Google's ad boxes and inject them into our system. If the user clicks on a Google ad (or if the ad is shown more than some threshold number of times) the client constructs a new Privad ad with the same contents, which it then sends to the broker to publish to other clients. The client synthesizes targeting and bid information since the scraped ad lacks both. The new ad is targeted to users that match some randomly selected subset of profile attributes of the user injecting the ad. Bids are currently random.

[2] `http://www.google.com/ads/preferences/view`

Showing Ads. Since we lack real publishers, we co-opt existing Google ad boxes to show Privad ads (which are re-injected Google ads). In order to not interrupt Google's business, we ensure that clicking on a Privad ad results in the same notification to Google as would have been sent had the original user that injected the ad clicked it.

Offload. The offload mechanism reduces broker load at the cost of client load, and must therefore be carefully balanced to not degrade the user's browsing experience. To this end, the client performs offloaded decryptions only when there is no user activity (e.g. mouse movements). We plan to additionally inhibit offload processing when the client computer is running on battery power, or is heavily loaded. To allow messages to be processed even when the offload client is unable to comply, the client generating the encrypted message encrypts it separately with the offload keys as well as the broker's public key as a fallback. The broker waits 2 minutes for offload clients to decrypt the message before performing the decryption itself.

7.1 Challenges

The primary implementation challenge is the effort required to scrape pages. Our implementation of the scraping modules for Facebook and Google Ad preferences comprises fully 20% of the client code. Adding additional websites, as well as keeping the modules updated with changes to websites is likely to require significant effort. Since webpage scraping is useful to a number of other addons and research projects, in the short-term we plan to crowd-source the development and maintenance of scrapers thus distributing the effort required. In the long-term, however, we envision designing tools that can generate much of the scraping code.

The second implementation challenge we face is defining the attribute hierarchy and mapping scraped information onto it. Currently we define the hierarchy as the superset of the information scraped — 9 demographic attributes (from Facebook) and 602 nested interest-based attributes (from Google) — which makes the task of mapping scraped information trivial. We cannot continue to do so, however, as we add more websites; Amazon alone, for instance, has 107K nested product categories. Since websites categories are slow-changing, however, one option is to generate a static mapping once and update that mapping as the website changes.

7.2 Pilot Deployment

We have deployed Privad with a small group of users comprised primarily of friends and family. The primary purpose of the deployment is convincing ourselves that Privad does not negatively impact users' browsing experience. As of this writing, 71 unique users (39% still active) have installed our addon based on anonymized status reports. Most users that uninstalled the addon did so within a day of installing. We believe this is because the addon currently offers little of value to the user. We are in the process of bundling other useful functionality to address this issue. We have not received any negative feedback from users[3].

Usage statistics from our deployment provide little insight — our userbase consists only of beta testers, and likely has a tech savvy bias owing to their ability to install a Firefox plugin. Nevertheless, we present some aggregate statistics for completeness. User profiles contained between 1 and 15 items (6 on average). Interestingly, 10 users deleted some information from their Facebook profile and Google Ad preferences page after, we believe, the addon prominently displayed to them the source of specific profile information. In the one month since we deployed Privad, the addon injected 165 unique ads scraped from Google. These resulted in a total of 1023 view reports, and 48 click reports. Note, however, the number of clicks is deceptively higher than we expected. We believe it is a result of our addon not taking the language of the ad into account; as a result, a number of Chinese and German ads were shown to a largely English-speaking audience who, no doubt, clicked to investigate. The broker was able to successfully offload 998 decryptions to clients (89% of those attempted); the rest failed due to clients going offline and were handled instead by the broker.

[3]or, for that matter, positive feedback

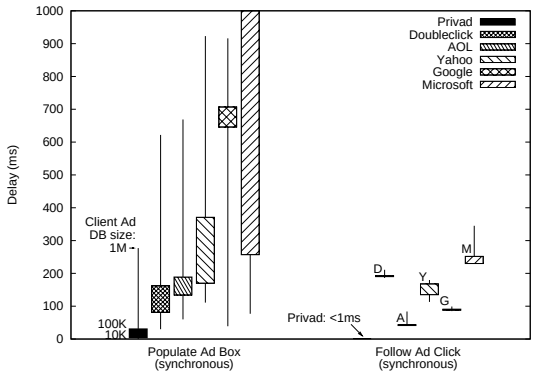


Figure 13: Privad eliminates network RTTs for showing ads, and reporting clicks. Whiskers for Privad show performance as the number of ads in the client's database scales to 1 million. Whiskers and boxes for existing ad networks show minimum and maximum latencies, and quartiles.

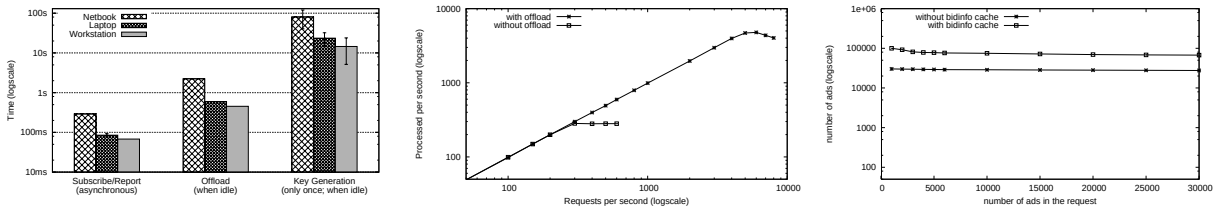
8 Evaluation

We use microbenchmarks to evaluate the scalability characteristics of our system. We lack datasets to perform meaningful macrobenchmarks or comparison studies.

8.1 Client Microbenchmarks

Eliminating Network RTTs. We first benchmark how Privad improves web browsing by eliminating network round-trips in the critical path of rendering webpages. Figure 13 compares Privad performance to existing ad networks. The figure compares the delay added for both populating ad boxes, and for fetching the advertiser webpage after a click. For Privad, we measured the time taken to populate ad boxes as we scale the number of ads cached in the client database by three orders of magnitude from 1K to 1M (shown as whiskers in the figure). Typically, however, we expect this number to be between 10K and 100K (shown as the box). For existing ad networks, we measure the time taken to fetch ad content, and the time taken to report a click and be redirected to the advertiser webpage. Since the tests are conducted from a German academic network, for fairness we measure performance only for popular German sites (based on Alexa rankings for websites ending in `.de`). We found existing ad networks use European servers for these websites, thus eliminating any unnecessarily cross-Atlantic round-trips. For existing networks, whiskers in the figure represent minimum and maximum delays encountered on 20 websites, and the boxes represents the first and third quartiles.

As one might expect, Privad outperforms existing networks since displaying ads requires only local disk access. In our implementation, the client stores ads in a SQLite database [7] with the necessary indexes to speed up the task of filtering ads based on keywords or webpage context. Even with 100K ads, which is an amount likely at the higher end of the number of ads matching a user's profile, Privad can populate ad boxes in 31ms. In existing networks, we found the delay was dominated by the ad selection process, by which we mean the time taken to fetch the generated iFrame object or Javascript that contains the URL of the ad content; downloading the ad content (up to 30kB for flash ads) took less than 2ms. Doubleclick, which to our knowledge does not perform demographic or context sensitive advertising, took 129ms in the median case, and Google, which does perform context sensitive advertising, took 670ms. Yahoo demonstrated bimodal behavior somewhat correlated with the type of ad (image vs. flash) ultimately served. We consistently noticed exceptionally high delays (several seconds) for Microsoft's ad network which, aside from ruling out DNS or packet-loss issues, we cannot explain. Overall, considering the prototype quality of our implementation, there appears to be much headroom for adding more complex ad selection strategies in Privad while still improving on the performance of existing ad networks.



(a) Client crypto overhead (b) Broker offload optimization (c) Broker auction performance

Figure 14: (a) Client overheads of public-key operations in Javascript are low enough to be masked with asynchronous reports, and idle-time processing. Privad protocol is designed with this in mind. (b) Offloading public-key operations improves broker performance significantly. (c) The envelop of auction performance at broker depending on privacy requirements.

With regards to reporting clicks, existing ad networks must perform a synchronous redirect where the browser first informs the ad network of the click before being redirected to the advertiser webpage. Some ad networks perform multiple redirects. Google, for instance, performs two synchronous redirects: first to a Google domain, and then to a Doubleclick domain, before redirecting to the advertiser; in the process, the browser sends both the Google cookie, and the Doubleclick cookie. Doubleclick performs three redirects. Synchronous redirects added between 100–200 ms before the advertiser website was even contacted. This represents up to 5% of the time considered the maximum threshold of acceptability for retail web page response times [1]. In contrast, Privad has virtually zero delay between the user clicking and the browser contacting the advertiser webpage since click reports are sent asynchronously in the background.

Public-Key Operations. Next, we benchmark public key operations in the client with an eye towards its impact on the user's browsing experience. Our key concern is that the Javascript execution model is single-threaded, and Firefox has no performance isolation between Javascript code in webpages, addons, and even that used to implement browser's user interface. As a result, a long running operation can cause the browser to freeze for the duration. Fortunately, as we show in Figure 14(a), overheads are low enough that they can be masked by deferring computations to brief periods of inactivity. In the figure, we compare the performance for three classes of clients: workstation, laptop, and netbooks. The workstation and laptop have a 3GHz and 2.1GHz Intel Core2 processor respectively, although Firefox, being single threaded, is limited to a single core. The netbook has a 1.6GHz single-core Intel Atom processor. We benchmark Firefox v3.5 that includes JIT optimizations for Javascript.

As shown in the figure, the workstation can construct subscription and view/click reports in 68ms. This includes one symmetric-key operation and three public-key encryptions (one with broker key, and two with offload keys); it takes 39ms with offload disabled. The laptop and netbook take 85ms and 160ms respectively. Since subscriptions and reports are generated asynchronously, in practice the overhead is imperceptible to the user.

Performing the offloaded decryption on behalf of the broker, however, is more expensive. The primary reason is that for RSA, public-key decryption is a factor of 10 more expensive than encryption (for our choice of 1024-bit keys). The workstation and laptop take around 0.5s to complete an offloaded decryption, and the netbook takes around 2s. Note, the broker can decrypt the same message in 3.6ms in Java (125x faster). Therefore by offloading to ten million idle clients, the broker can potentially boost its effective processing capability by about 80K cores while saving on datacenter and cooling costs. The client also incurs a one-time cost of generating the public-private key pair used for offload. Since generating public keys requires a search for large primes, it takes significantly longer (on average, 14s on the workstation, 80s on the netbook). While we currently mask this by waiting for 15 minutes of inactivity, using native code in the client or generating the key pair once at the broker and securely transmitting it to the client is an option.

8.2 Broker Microbenchmarks

We benchmark the broker on a dual-core 64-bit 3GHz Dell Optiplex 760 workstation with 4GB RAM. We use the loopback network interface to eliminate network bottlenecks. We limit the JVM to a single-core and 1GB memory to isolate the broker from the benchmark tool. In developing the broker, we opted to reuse the text-based JSON RPC framework for broker-dealer messages (incorrectly) assuming that cryptographic costs would dwarf RPC overheads; in retrospect this was a mistake. With the offload optimization we reached a point where RPC serialization and deserialization accounts for 43% of processing time. Consequently, the numbers we report below are lower bounds. We also report performance without RPC overhead for an upper bound.

Subscribe/Report. We first focus on the performance of subscribe and report messages at the broker since they involve public-key operations. Figure 14(b) plots broker performance for subscribe and report messages, with and without the offload optimization. Messages are typically 750 bytes long. Without offload, as expected, performance is bottlenecked by RSA decryptions at around 280 decryptions per second (on a single-core). Offloading decryptions improves performance by a factor of 20. Beyond 6K requests per second, the bottleneck is primarily due to RPC overhead. The broker can otherwise perform 33K raw AES decryptions per second once the offload client recovers the AES key. Note with hardware AES support this limit can be increased still further.

Subscribe and report messages additionally affect state at the broker. Our implementation can handle 1.8M subscriptions before exhausting the allotted 1GB memory. Reports do not consume memory since they modify bookkeeping state in-place. They do, however, consume disk space (24 bytes per log message) but only for click-reports. Given the typically low ratio of clicks to views, disk throughput or capacity is not a big concern.

Publish. Our broker can publish 8.5K ads per second (734M per day). This number almost doubles if RPC overhead is excluded, at which point performance is bottlenecked by the cost of encrypting the ad with the client-supplied AES key.

Auction. Processing auctions presents a tradeoff. First, dealer decides the number of ads that the broker must rank. The more ads in a message, the better the privacy properties of the mix, while the fewer ads in the message, the faster the sort. Second, the broker decides whether or not to cache the opaque (encrypted) bid information across auctions. Not recomputing the opaque bid improved performance, but has the potential to leak information to the dealer if advertiser bids are unique. Figure 14(c) plots broker performance as we vary the number of ads in the auction, and enable or disable opaque bid caching. With cached opaque bids, sorting dominates processing, with a 30% throughput drop for large auctions. AES encryption costs dominate when the opaque bids are regenerated, lowering auction throughput to around 30K ads per second. In either case, since auction throughput exceeds that of ads published, it is not a scalability concern.

For both publish and auction messages, we found performance does not depend on the number of unique ads or clients since all lookups are $\mathcal{O}(1)$. Note our broker keeps all ads in memory; in a full deployment, ads will be stored in a database and cached at the broker, which could result in different bottlenecks.

8.3 Dealer Microbenchmarks

We benchmarked the dealer on the same hardware as the broker. Dealer performance is bottlenecked by the number of client HTTP connections. Clients poll at 30s intervals (configurable) to retrieve ads published for them. Our dealer can handle 6.5K connections per second; thus we estimate a single dealer instance can support nearly 200K online clients. Our dealer can additionally forward 15K messages per second in either direction (clients to broker, or vice versa). This is far lower than we were expecting. The bottleneck is the text-based RPC protocol, the parser for which saturates at 7MBps per core.

8.4 Evaluation Summary

Overall, the performance of Privad mechanisms is independent of the number of clients and ads. We have been able to optimize all parts of the system to a point where our original choice of RPC protocol is now the limiting bottleneck. The next step in scaling then is to explore more streamlined wire protocols.

That said, microbenchmarks, as comprehensive as they might be, give only a partial answer — that no single aspect of our system or implementation fundamentally limits scalability. We leave many questions unanswered. What is the overhead of publishing non-matching ads for real user profiles and targeting information? How many ads and for how long should they be cached at the client? How much past history do the dealer and broker need to remember to defend against click-fraud? How does better targeting affect the ad economy? We cannot answer these questions without public datasets for Internet scale advertising systems. Data about targeting and bid information for ads, volume and rate of change of ads, user profiles and view/click traces, and real attacks and their economic impact would go a long way towards answering these questions. Nevertheless, given our prototype implementation and microbenchmarks, we believe the basic scalability argument for Privad is on a solid footing.

9 Related Work

There is very little past work on the design of private advertising systems, and what work there is tends to focus on isolated problems rather a complete system like Privad. Juels [21] focuses on the private ad dissemination problem. Juels opts for a distributed mixnet design. The dealer in Privad, in contrast, is simpler and more scalable, and unlike an anonymizing mixnet, is in position to help the broker defend against click-fraud. In Databank [25], the authors propose a radically new economic model where advertisers pay clients to access their private information, and use market incentives to preserve privacy. Privad provides stronger privacy guarantees while operating within the existing economic model.

Existing research on defending against click-fraud is complementary to Privad. Juels *et al.* [22] propose the notion of "Premium Clicks" for authenticated users; this is easily done at the dealer. It can additionally be combined with robot detection through activity tacking [29] and IP blacklisting [19]. At the same time, the broker can use the statistical learning techniques proposed in [18] unchanged.

Auctions are another area where past work is largely complementary to Privad. In general, however, making these auctions private is not straightforward, as illustrated by our construction that adapts GSP auctions [10]. Nevertheless, we believe principles behind Hybrid auctions [12], Combinatorial auctions [33] and auctions with frequency-capping [11] can be incorporated in Privad.

Preserving privacy in general is an area of active research. Private Information Retrieval enables querying a database without letting the database learn what was queried. [27] presents a survey. As with Juels' ad dissemination system, these tend to be overkill, scale less well, and make click-fraud prevention impossible. Differential Privacy [9] allows statistical databases to be mined for aggregate statistics without compromising an individual's data in the database. TOR [8] protects user privacy on the web. All three lines of research target general problems, and as a result, strive to provide privacy guarantees that hold in the broadest of settings. By restricting the problem domain to online advertising with well-defined players, we are able to construct a simpler and more scalable solution.

10 Summary and Future Work

This report describes a practical private advertising system, Privad, which attempts to provide substantially better privacy while still fitting into today's advertising business model. There are many major components to such a system: ad delivery and reporting, click fraud defense, user profiling, advertiser auctions, and post-click anonymization. While we have designs and detailed privacy analysis for all of these components, we have solid performance results only on the ad delivery and reporting components, and even here we have not tested the system to anywhere near scale. The rest of the system still requires substantial experimentation, much of which can only be done with either real users or real trace data (i.e. from existing advertising systems).

Our next steps, then, are to do this experimentation, as well start a dialogue with privacy advocacy groups to better understand if our privacy is "good enough".

References

[1] Akamai Technologies, Inc. Akamai and Jupiter Research Identify '4 Seconds' as the New Threshold of Acceptability for Retail Web Page Response Times. `http://tinyurl.com/ydymfs`, Nov. 2006.

[2] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. IETF Request For Comments (RFC 4033), Mar. 2005.

[3] R. Baden, A. Bender, D. Starin, N. Spring, and B. Bhattacharjee. Persona: An Online Social Network with User-Defined Privacy. In *Proceedings of the 2009 Conference of the Special Interest Group on Data Communication (SIGCOMM)*, Barcelona, Spain, Aug. 2009.

[4] H. Ballani, P. Francis, and X. Zhang. A Study of Prefix Hijacking and Interception in the Internet. In *Proceedings of the 2007 Conference of the Special Interest Group on Data Communication (SIGCOMM)*, Kyoto, Japan, Aug. 2007.

[5] J. Chester, S. Grant, J. Kelsey, J. Simpson, L. Tien, M. Ngo, B. Givens, E. Hendricks, A. Fazlullah, and P. Dixon. Letter to the House Committee on Energy and Commerce. `http://tinyurl.com/y85h98g`, Sept. 2009.

[6] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. *Journal of the ACM*, 45:965–981, Nov. 1998.

[7] D. Richard Hipp and Dan Kennedy and Shane Harrelson and Christian Werner. SQLite Home Page. `http://www.sqlite.org`.

[8] R. Dingledine, N. Mathewson, and P. Syverson. TOR: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium (Security '04)*, San Deigo, CA, Aug. 2004.

[9] C. Dwork. Differential Privacy: A Survey of Results. *Theory and Applications of Models of Computation*, 4978:1–19, 2008.

[10] B. Edelman, M. Benjamin, and M. Schwarz. Internet Advertising and the Generalized Second-Price Auction: Selling Billions of Dollars Worth of Keywords. *American Economic Review*, 97(1):242–259, Mar. 2007.

[11] A. Farahat. Privacy Preserving Frequency Capping in Internet Banner Advertising. In *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*, pages 1147–1148, Madrid, Spain, 2009.

[12] A. Goel and K. Munagala. Hybrid Keyword Search Auctions. In *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*, pages 221–230, Madrid, Spain, 2009.

[13] Google Inc. Google Web Toolkit. `http://code.google.com/webtoolkit`.

[14] G. Gross. FTC Sticks With Online Advertising Self-regulation. *IDG News Service*, Feb. 2009.

[15] S. Guha, B. Cheng, A. Reznichenko, H. Haddadi, and P. Francis. Privad: Rearchitecting Online Advertising for Privacy. Technical Report TR-2009-4, Max Planck Institute for Software Systems, Kaiserslautern- Saarbrücken, Germany, 2009. `http://mpi-sws.org/tr/2009-004.pdf`.

[16] S. Guha, A. Reznichenko, H. Haddadi, and P. Francis. Serving Ads from `localhost` for Performance, Privacy, and Profit. In *Proceedings of the 8th Workshop on Hot Topics in Networks (HotNets '09)*, New York, NY, Oct. 2009.

[17] S. Guha, K. Tang, and P. Francis. NOYB: Privacy in Online Social Networks. In *Proceedings of The First ACM SIGCOMM Workshop on Online Social Networks (WOSN '08)*, Seattle, WA, Aug. 2008.

[18] N. Immorlica, K. Jain, M. Mahdian, and K. Talwar. Click Fraud Resistant Methods for Learning Click-Through Rates. In *Proceedings of the 1st International Workshop on Internet and Network Economics (WINE '05)*, Hong Kong, China, Dec. 2005.

[19] B. J. Jansen. Adversarial Information Retrieval Aspects of Sponsored Search. In *Proceedings of the 2nd Workshop on Adversarial Information Retrieval on the Web (AIRWeb '06)*, Seattle, WA, 2006.

[20] A. Jesdanun. Ad targeting based on ISP tracking now in doubt. In *Associated Press*, Sept. 2008.

[21] A. Juels. Targeted Advertising ... And Privacy Too. In *Proceedings of the 2001 Conference on Topics in Cryptology*, pages 408–424, London, UK, 2001. Springer-Verlag.

[22] A. Juels, S. Stamm, and M. Jakobsson. Combating Click Fraud via Premium Clicks. In *Proceedings of 16th USENIX Security Symposium (Security '07)*, pages 1–10, Boston, MA, 2007.

[23] B. Krishnamurthy and C. Wills. On the Leakage of Personally Identifiable Information Via Online Social Networks. In *Proceedings of The Seconds ACM SIGCOMM Workshop on Online Social Networks (WOSN '09)*, Barcelona, Spain, Aug. 2009.

[24] B. Krishnamurthy and C. E. Wills. Cat and Mouse: Content Delivery Tradeoffs in Web Access. In *Proceedings of the 15th international conference on World Wide Web (WWW '06)*, Edinburgh, Scotland, 2006.

[25] R. M. Lukose and M. Lillibridge. Databank: An Economics Based Privacy Preserving System for Distributing Relevant Advertising and Content. Technical Report HPL-2006-95, HP Laboratories, 2006.

[26] R. Miller. Keynote: Ad Networks Failed, Not News Sites. `http://tinyurl.com/ychd3rn`, June 2009.

[27] R. Ostrovsky and W. E. S. III. A Survey of Single-Database Private Information Retrieval: Techniques and Applications. *Public Key Cryptography*, 4450:393–411, 2007.

[28] K. Park, V. S. Pai, K.-W. Lee, and S. Calo. Securing web service by automatic robot detection. In *Proceedings of the USENIX Annual Technical Conference*, Boston, MA, USA, May 2006.

[29] K. Park, V. S. Pai, K.-W. Lee, and S. Calo. Securing Web Service by Automatic Robot Detection. In *Proceedings of the 2006 USENIX Annual Technical Conference*, Boston, MA, 2006.

[30] The Eclipse Foundation. Jetty. `http://www.eclipse.org/jetty`.

[31] `http://www.dmoz.org/`. Dmoz: Open directory project.

[32] Versaneo GmbH. pidCrypt - pidder's JavaScript crypto library. `http://www.pidder.com/pidcrypt`.

[33] S. D. Vries and R. V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, 15(3):284–309, 2003.

[34] P. R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, 1995.