Technical Report: MPI-SWS-2010-001

Privad: Practical Privacy in Online Advertising

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

ABSTRACT

Online advertising is a major economic force in the Internet today. Today's deployments, however, erode privacy and degrade performance as browsers wait for ad networks to deliver ads. We present Privad, a practical private online advertising system. Privad serves ads from the endhost; this is attractive from three standpoints — privacy, profit, and performance. Tracking the user's profile on their computer and not at a third-party improves privacy. Better targeting and potentially lower operating costs can improve profits. And relying more on the local endhost rather than a distant central third-party improves performance. We have implemented Privad and deployed it on a small scale. This paper focuses on the scalability aspects of Privad. It describes the Privad architecture and protocols, analyzes its scalability characteristics, and presents measurements that substantiate our scalability claims.

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 [21] and poor user experience [23]. And while privacy advocates would like to put an end to advertising models that violate privacy, aside from a few highly publicized battles [17], they have had little success with the more entrenched ad brokers like Google and Yahoo! [12]. Arguably the reason why privacy advocates have failed is that they offer no viable alternatives. The deal they offer, privacy *or* personalization [2, 30], is not acceptable to the entrenched players. This paper takes a first stab at providing that alternative.

In this paper, we present 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 [20].

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 [3]. 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 roundtrips 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 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.

The contributions of this paper are as follows: It presents what is to our knowledge the first arguably *practical* private advertising system. It describes the design and implementation of Privad, and gives microbenchmark performance results for the implementation. This paper focuses on the scalability aspects of Privad. While it does provide a brief overview of Privad's privacy characteristics, it does not provide a full analysis of Privad's privacy. This can be found in [13]. Our implementation does some simple user profiling, disseminates ads, runs auctions, displays ads, reports views and clicks, and contains a simple click-fraud defense mechanism. While we study the scalability of these mechanisms, we do not study their quality or effectiveness. This is left for future work.

We readily acknowledge that user profiling is not the only privacy issue that plagues the Internet, or even the most important (identity theft comes to mind). User profiling is, however, an important problem, and one that is not isolated from other privacy issues. For instance, arguably the primary motivation for social networks to gather private and Personally Identifying Information (PII) is ultimately in support of advertising. Overall, Privad, along with its proof-of-concept implementation and pilot deployment, represents an argument that highly-targeted practical online advertising and good user-privacy are not mutually exclusive. We hope that this first stab at a feasible design leads to additional research on privacy in advertising as well as on privacy in other aspects of online life.

2. PRIVAD OVERVIEW

2.1 Model

There are five players in Privad: user/client, publisher, advertiser, broker, and dealer. User, publisher, advertiser, and broker all have analogs in today's advertising model, and play the same basic business 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.

2.2 Privad Operation

Figure 1 illustrates an overview of Privad oper-

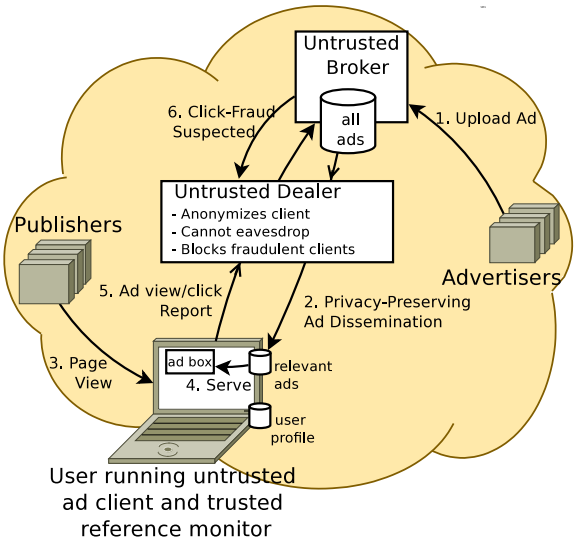


Figure 1: The Privad architecture

ation. 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.tennis.racket` or `outdoor.lawn-care`. Demographics include things like gender, age, salary, and location.

Although good targeting is one of the main requirements for Privad, we have not shown that this is the case. As compared to the shallow view into the behavior of a vast number of users that current brokers like Google have, the Privad client has a much deeper view into any given user. We suspect that relatively simple techniques like watching the user's shopping activity, scraping the user's social network profiles, seeing what applications the user runs or what music the user listens to, and what websites the user spends time on, can go a long ways towards targeting the users interests and demographics. Future work will determine if the deeper view of Privad is enough to match or exceed the broader view.

Advertisers upload ads to the broker, including the set of interests and demographics targeted by each ad, and optionally, search or website keywords. The broker distributes some fraction of these 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 some matching 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.

Ad auctions determine both which ads are shown to the user and in what order. In addition to bid information, ranking is based on both user and global metrics. User metrics include things like how well the targeting information matches the user, and the user's past interest in similar ads. Global metrics include the aggregate click-through-rate observed for the ad, the quality of the advertiser webpage, etc.

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 from the local database based on keywords or webpage context 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 mitigation strategy is for the dealer to suppress reports from attacking clients. There are two classes of mechanisms for identifying attacking clients — by the dealer alone, and by the broker in conjunction with the dealer. For instance, the dealer may identify an attacking client directly when the client transmits too many reports or subscription requests. Or, the broker may use statistical approaches (e.g. [15]) to identify which publishers or advertisers are under attack, and indicate to the dealer which reports relate to these attacks. The dealer then traces these reports back to the clients responsible. Any clients associated with a threshold number of attacks are identified as attackers.

2.3 Client Framework

Messages between client and broker are encrypted so that the dealer cannot see their contents. It is critical, however, that users, 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 Pri-

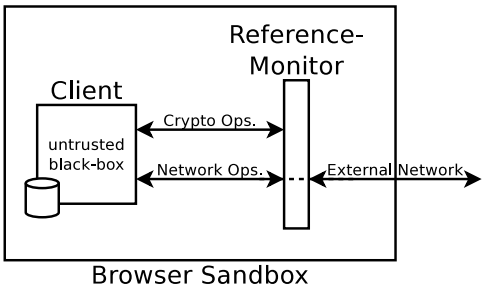


Figure 2: The Client framework

vad client architecture allows for a reference monitor (Figure 2). The reference monitor framework allows any third party software trusted by the user to interpose itself between the client and the external world. The reference monitor validates message contents and performs encryption operations, and ensures the content of outgoing messages matches expectations. This software can insure that Privad is operating according to design. This shifts trust from the Privad client to the third-party software, 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. These clients could operate independently of each other, for instance with each client fully implementing the Privad protocol, scraping webpages, and even arranging for separate dealers. Alternatively, there could be some common support in the user's browser to handle multiple clients more efficiently, for instance, by sharing a common Privad protocol implementation and common webpage scraping modules. Multiple brokers could also share dealers.

3. PRIVAD DETAILS

This section provides details on ad dissemination, auctions, view/click reporting and click-fraud defense, the reference monitor, and the offloading of public-key cryptographic operations. It also puts forth some of the rationale for our design decisions. These details represent a snapshot of our current thinking. While reporting, ad dissemination, the reference monitor, and crypto-offloading are quite stable, the click-fraud defense, and auctions may easily evolve as we do more analysis and testing. We provide them here so as to present a complete argument for Privad's viability.

3.1 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 [14], we concluded that there are too many ads and 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 no sensitive information is revealed in the subscriptions, there are a large number of users with the same subscription (k-anonymity), 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 3). 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, 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.

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

Figure 3: 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.

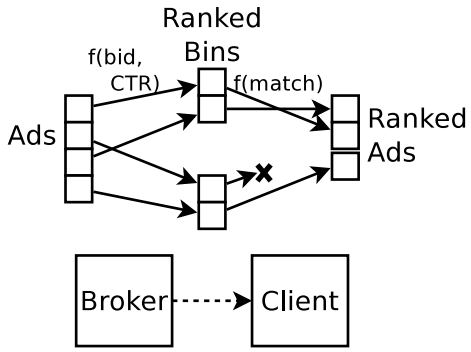


Figure 4: Design-I: Simple Auctions. For each Pub-Sub channel, broker bins ads by bid and global click-through rate. For each bin, client ranks ads by quality of match, filtering ads that don't match the user.

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.

3.2 Ad Auctions

Auctions determine which ads are shown to the user and in what order. The goal of the auction is to provide a fair marketplace where advertisers can influence the frequency and position of their ads through their bids. The challenge, of course, is in doing so while preserving privacy of the user, as well as the advertiser's bid. As a proof of viability, we present two auction designs that meet our requirements. The first design implements a basic auction. The second design, which is more complex, effectively implements the GSP auction used by Google today [8] within the confines of the Privad model. Other privacy-preserving auction designs may be possible, and are for further study.

3.2.1 *Design-I: Simple Auctions*

The simplest approach is for the broker and client to conduct auctions at ad dissemination time (Figure 4). 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 three times 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 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.

3.2.2 *Design-II: Combined Auctions*

In the combined approach (Figure 5), 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

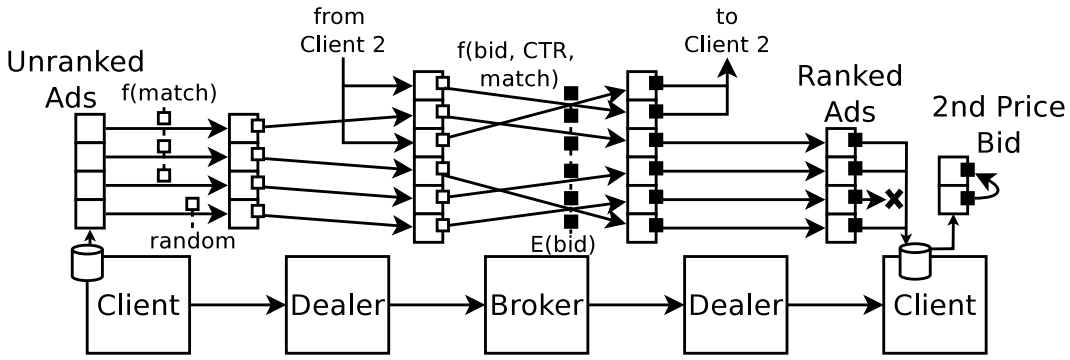


Figure 5: Design-II: Combined Auctions. Client annotates ads (across all channels) with quality of match, or random number if the ad doesn't match the user. Dealer mixes annotations from multiple clients. Broker ranks ads by bid, global click-through rate, and match quality, and annotates the result with opaque bid information. Dealer slices auction result by client. Client filters out non-matching ads. Client reports second-price bid on click.

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 them to the clients. 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 *Iid*s are single-use, they do not leak ad information to the dealer.

The issue of volatility is solved using second-price auctions. 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; see [13] for details and a privacy analysis.

Overall, the combined auction effectively duplicates the GSP auction used by Google [8] without sacrificing client or bid privacy.

3.3 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 6). 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. 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.

3.4 Click-Fraud Defense

Neither Privad nor current brokers have a silver bullet against click-fraud. We provide a basic mechanism for anonymously identifying defrauding clients. When the broker detects that click-fraud is happening (either on an advertiser or publisher), it notifies the dealer of the *Rid* of reports related to the advertiser or publisher. If the dealer receives a thresh-

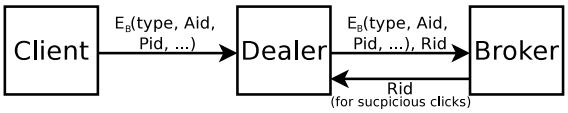


Figure 6: Message exchange for view/click reporting and blocking click-fraud. B is the public key of the broker. *Aid* identifies the ad. *Pid* identifies publisher website or application where the ad was shown. For second-price auctions, the opaque auction result is included. *Rid* uniquely identifies the report at the dealer.

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 1: Reference monitor API

old number of notifications for a given client, then that client's subsequent reports are tagged as suspicious. The dealer can also itself monitor clients to see if they have an unusually high volume of views or clicks, and tag them accordingly.

Detecting click-fraud at the broker will require a variety of techniques that will evolve over time as an arms race between broker and attacker. These techniques include keeping historical statistics on click and view volumes and looking for anomalies, building honey farms that attract and identify click-fraud malware, and dealers sharing black-lists of attacking clients. One interesting technique that we have designed is what we call a "bait ad": a kind of captcha for ads. Bait ads contain the targeting information of one ad, but the text of a very different ad. If a click-fraud is suspected, the broker can start sending out some bait ads. An unusually high rate of clicks on bait ads helps verify that an attack is underway and helps identify the fraudsters. A more detailed description of bait ads and other click-fraud detection mechanisms can be found in [13].

3.5 Reference Monitor

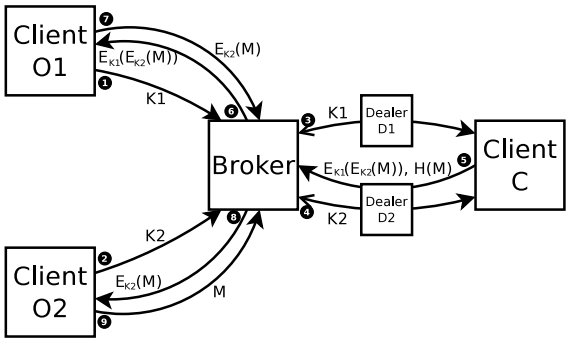
Table 1 lists the complete API exposed by the reference monitor. The client uses the monitor's ENCRYPT function to encrypt all messages. Additionally, the client uses the monitor's GENSYMMETRICKEY functions to generate symmetric keys for subscribe messages. The reference monitor validates message contents 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 the randomized padding in the encrypted message).

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.

3.6 Offloading Public-Key Operations

We present an optimization to Privad that reduces broker overhead by leveraging idle clients to perform pubic-key operations. Figure 7 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 $O1$, 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 $O2$, which responds with the original plain-text M. The broker validates the decryption by computing the hash of M and comparing it to



#	From	To	Message
1	Client $O1$	Broker	$K1$
2	Client $O2$	Broker	$K2$
3	Broker	Client C (request-response via $D1$)	$K1$
4	Broker	Client C (via $D2$)	$K2$
5	Client C	Broker (via $D2$)	$E_{K1}(E_{K2}(M)), H(M)$
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 7: Offloading public-key operations to idle clients. $K1$ and $K2$ are public-keys created by clients $O1$ and $O2$ respectively. $E_x(M)$ represents the encryption of M under key x, and $H(M)$ represents a cryptographic hash of M. 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 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 [13].

4. IN BRIEF: USER PRIVACY

In this section, we provide a high-level overview of the privacy properties of Privad. A detailed security analysis can be found at [13]. Of the system players (user/client, broker, dealer, publisher, and advertiser), we are primarily interested in user privacy. There are also advertiser privacy concerns (targeting, bidding, and budget information), but due to lack of space they are not discussed here.

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. While it appears possible for Privad to provide user anonymity and prevent this sort of exploitation [13], space prevents discussion of it in this paper. The user privacy analysis provided here, then, is limited to user interactions with the broker.

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 [4], 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 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 [13]. 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 monitor can be verified by hand.

5. IMPLEMENTATION

We have implemented the full Privad system. The system comprises a client and reference monitor implemented as a 154KB addon[1] 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) [11] 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 3.2.2, and the offload optimization described in Section 3.6 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 [28] 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, which is more compact than XML RPC. The broker and dealer are written as Java servlets, hosted by the Jetty webserver [27]. The broker uses standard Java 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.

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 activ-

[1] `http://adresearch.mpi-sws.org/addon.html`

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

ity (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.

5.1 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.

5.2 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].

[3]or, for that matter, positive feedback

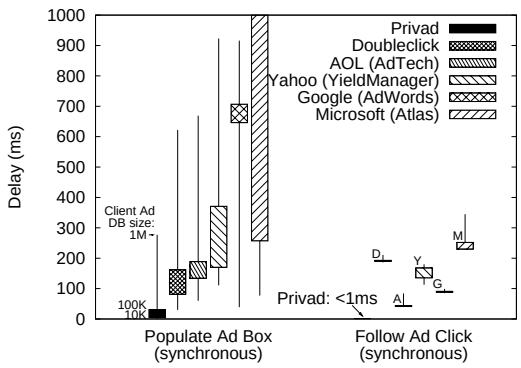


Figure 8: Privad eliminates network RTTs for showing ads, and reporting clicks. Whiskers for Privad show performance as the number of ads in the client's database scales to 1 million. Whiskers and boxes for existing ad networks show minimum and maximum latencies, and quartiles.

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.

6. EVALUATION

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

6.1 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 8 compares Privad per-

formance 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 roundtrips. 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.

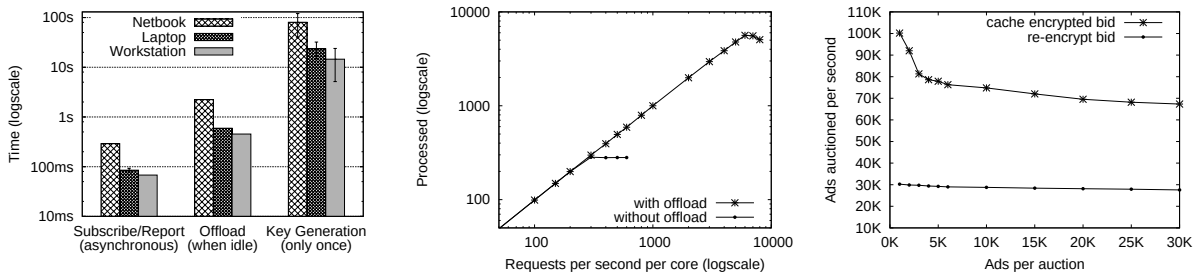
As one might expect, Privad outperforms existing networks since displaying ads requires only local disk access. In our implementation, the client stores ads in a SQLite database [5] with the necessary indexes to speed up the task of filtering ads based on keywords or webpage context. Even with 100K ads, 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.

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 9(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



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

Figure 9: (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.

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.

6.2 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 9(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 9(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 subscriptions or unique ads since all lookups are $\mathcal{O}(1)$. Note our broker keeps all ads in memory; in reality, ads will be stored in a database and cached at the broker, which could result in different bottlenecks.

6.3 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.

6.4 Evaluation Summary

Overall, Privad mechanisms appear to scale to large numbers 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.

Microbenchmarks, not matter how comprehensive, 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.

7. 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 [18] 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; Privad shares this insight with [26], where Ringberg *et al.* use a dealer-like mechanism to solve a different problem — scalable private data aggregation. Furthermore, unlike an anonymizing mixnet, the dealer is in position to help the broker defend against click-fraud. In Databank [22], 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.* [19] 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 [25] and IP blacklisting [16]. At the same time, the broker can use the statistical learning techniques proposed in [15] 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 [8]. Nevertheless, we believe principles behind Hybrid auctions [10], Combinatorial auctions [29] and auctions with frequency-capping [9] 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. [24] 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 [7] allows statistical databases to be mined for aggregate statistics without compromising an individual's data in the database. TOR [6] 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.

8. SUMMARY AND FUTURE DIRECTIONS

This paper 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, adver-

tiser 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".

9. 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. 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.

[3] 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.

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

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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`.

[14] S. Guha, A. Reznichenko, K. Tang, 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.

[15] 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.

[16] 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.

[17] A. Jesdanun. Ad Targeting Based on ISP Tracking Now in Doubt. *Associated Press*, Sept. 2008.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] H. Ringberg, B. Applebaum, M. J. Freedman, M. Caesar, and J. Rexford. Collaborative, Privacy-Preserving Data Aggregation at Scale. Cryptology ePrint Archive, Report 2009/180, 2009. `http://eprint.iacr.org`.

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

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

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

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