PRIVATE-BY-DESIGN ADVERTISING AND ANALYTICS: FROM THEORY TO PRACTICE

Technical Report MPI-SWS-2014-005

Alexey Reznichenko

Max Planck Institute for Software Systems (MPI-SWS)

June 16, 2014

PRIVATE-BY-DESIGN ADVERTISING AND ANALYTICS: FROM THEORY TO PRACTICE

A Dissertation

Presented to the Faculty of the Computer Science Department

of the Technical University Kaiserslautern

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Engineering (Dr.-Ing.)

by

Alexey Reznichenko

April 2014

ABSTRACT

There are a number of designs for an online advertising system that allow for behavioral targeting without revealing user online behavior or user interest profiles to the ad network. Although these designs purport to be practical solutions, none of them adequately consider the role of ad auctions, which today are central to the operation of online advertising systems. Moreover, none of the proposed designs have been deployed in real-life settings. In this thesis, we present an effort to fill this gap. First, we address the challenge of running ad auctions that leverage user profiles while keeping the profile information private. We define the problem, broadly explore the solution space, and discuss the pros and cons of these solutions. We analyze the performance of our solutions using data from Microsoft Bing advertising auctions. We conclude that, while none of our auctions are ideal in all respects, they are adequate and practical solutions. Second, we build and evaluate a fully functional prototype of a practical privacy-preserving ad system at a reasonably large scale. With more than 13K opted-in users, our system was in operation for over two months serving an average of 4800 active users daily. During the last month alone, we registered 790K ad views, 417 clicks, and even a small number of product purchases. Our system obtained click-through rates comparable with those for Google display ads. In addition, our prototype is equipped with a differentially private analytics mechanism, which we used as the primary means for gathering experimental data. In this thesis, we describe our first-hand experience and lessons learned in running the world's first fully operational "private-by-design" behavioral advertising and analytics system.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 On the Internet Someone Knows You Are a Dog

Third-party tracking of users is widespread and increasing [43]. One of the primary purposes of user tracking is behavioral advertising. Although many companies that participate in tracking and behavioral profiling claim to not gather Personally Identifying Information (PII), it is often easy to link tracking information with PII [42]. While tracking has been going on for a number of years, the public awareness of this as a privacy problem has recently skyrocketed. At the same time, literally dozens of behavioral targeting companies have emerged, all vying with each other for the number and quality of data sources, and for their ability to target users based on this data. Public awareness of online privacy erosion has already had some direct impact on the advertising ecosystem: recently the start-ups Phorm and NebuAd were heavily criticized for their targeting practices and sloppy opt-in or opt-out policies [39]. Arguably most objections were triggered by the threat of "wiretapping" and "deep packet inspection", but nevertheless these concerns were enough to cancel or delay pilot deployments and cause serious damage to the reputations of these companies. Today, we are approaching the same level of concern over online tracking, leading, for instance, to the Do-Not-Track (DNT) initiative launched by the Federal Trace Commission (FTC) and to proposals for a Do-Not-Track registry [56]. However, DNT provides only an illusion of privacy while destroying targeting utility, and as a result has come to stand for "Do-Not-Target". Not surprisingly,

DNT has not gained a lot of traction with the public, and ad networks are actively ignoring it.

1.2 Private-by-Design Advertising

Several research projects have proposed "private-by-design" behavioral advertising systems (Adnostic [58], RePriv [29], Privad [34], MobiAd [36], and PiCoDa [49]). Rather than simply opting out of tracking, or pushing for data-protection after the fact, they argue that privacy can be embedded from the start and propose alternative advertising technologies that allow for behavioral targeting without revealing user online behavior or user interest profiles to the ad network.

Although the existing private-by-design advertising systems differ in significant ways, they all share several key design components. All systems propose a *software agent* that runs at the client and generates a user profile. All systems at least share the privacy goal that this profile not be revealed. All these designs propose that the broker transmit multiple ads to the client, not all of which match the user profile. For instance, the ads may all be within a given interest category. From among these ads, the client then locally selects the ads which best match the user profile, and displays these to the user. The client reports the result of this selection anonymously, and without letting the broker link together different components of the user profile. The key privacy mechanisms are therefore *anonymity* and *unlinkability*.

The private-by-design ad systems are meant to be realistic alternatives that industry finds attractive. None of these systems, however, adequately explore

how to operate the auctions that are critical to current advertising systems. Without this component, these systems leave unanswered what revenue the *broker* (i.e. an ad network like Google) can earn, thereby reducing the likelihood that a private-by-design advertising system will be of commercial interest. In this thesis, we address the challenge of running auctions that leverage a *user profile* for ad ranking while keeping the user profile private.

Each of the systems cited above claims to be practical in that they provide both good privacy and good utility at reasonable cost. However, none of the proposed designs have been deployed and evaluated in real-life settings. This thesis attempts to fill this gap and answer a number of fundamental questions: "Is private-by-design advertising really practical?", "Can relevant ads be delivered?", "Is it possible to adequately measure the systems without infringing on user privacy?", "Will unforeseen devils in the details kill the system?" To do so, we built, deployed, and evaluated a fully functional prototype of a private-by-design ad system based on the Privad design [33]. Our deployment delivered functional ads in the sense that the ads were targeted to user interests, displayed on publisher webpages, linked to real shopping websites, and in fact led to actual purchases. Side-by-side with Privad, we also deployed a distributed differentially-private user analytics system, PDDP [19], that served as our primary means of gathering experimental data.

1.3 Contributions

Altogether, this thesis makes the following contributions towards transforming private-by-design advertising from a theoretical abstraction to a practical sys-

tem that both fits within the current economical model and is a demonstrably viable alternative to the current non-private designs:

- It proposes two new private-by-design auction mechanisms, and a third based on the previous work but substantially improved.
- It analyzes the trade-offs between these three designs in terms of privacy properties, auction properties, and fraud resistance.
- It analyzes the effect of bid churn and auction timing on revenue and ad ranking using a trace of Bing search advertising auctions, and uses this analysis to argue for the feasibility of the solutions.
- It presents the design and analysis of the first deployed fully-functional private-by-design ad system. In so doing it preliminarily shows that private-by-design behavioral advertising is practical.
- It presents our experience in running a relatively large-scale user-centric research experiment with differentially private analytics as the primary means of gathering system and user data. It shows that such an approach is feasible.
- It analyzes the practical implications of the privacy deficits accumulated as a result of differentially private data collection. It concludes that differential privacy is a poor model for understanding privacy loss in our deployment (too pessimistic), and that its noise-adding mechanism alone is too weak to be practical.

Some of the material presented in this thesis was previously published in a technical report [33], a series of conference papers [34, 54, 19], or is under submission to a conference at the time of this writing [53].

1.4 Structure of this Thesis

This thesis has two parts. The first part addresses the challenges of running private-by-design auctions that leverage user profiles for ad ranking while keeping the user profile private. It defines the problem, broadly explores the solution space, and discusses the pros and cons of the proposed solutions. The second part presents an effort to build and evaluate a fully functional prototype of a practical private-by-design ad system at a reasonably large scale.

Part I contains Chapters 2 through 5. Chapter 2 provides background on online ad auctions in the current advertising systems. It also outlines an abstract alternative advertising model that captures key aspects of the existing proposals for private-by-design advertising. Chapter 3 describes the design goals of the auction component in the private-by-design advertising system. It then proposes there solutions and discusses each in detail. Chapter 4 analyzes the three auction designs in terms of privacy, auction quality, and attacks on the auction component. Chapter 5 analyzes the performance of the proposed solutions using data from Microsoft Bing advertising auctions.

Part II contains Chapters 6 through 9. Chapter 6 gives a broad overview of the Privad architecture and discusses the privacy guarantees provided by the system. It also outlines the design of the PDDP system, which enables differentially private data collection in distributed settings. Finally, it describes previous attempts at building and deploying a private-by-design advertising system. The experimental Privad prototype we have built is described in detail in Chapter 7. Chapter 8 describes our experience in deploying the Privad prototype and presents results obtained during the deployment. Chapter 9 explores

the extent to which a differentially private data collection system can be used to understand what is going on behind the scenes in the private-by-design ad deployment. It also looks at the privacy deficits accumulated as a result of our analysis and studies the privacy implications for the end users.

Chapters 1 and 10 fall outside of the two Parts. Chapter 1 is the introduction. Chapter 10 concludes the thesis and outlines directions for future work.

Part I

Auctions in Private-by-Design Internet Advertising

CHAPTER 2

BACKGROUND AND RELATED WORK

In order for the private-by-design advertising to become a viable alternative to the current tracking systems, it has to operate within the existing business model. In particular, it must preserve the auction mechanism that allows advertisers to compete for ad slots and also determines revenues generated by the ad networks.

The most common pricing model for online advertising systems today is Pay Per Click[1] (PPC): the advertiser does not pay the broker for showing an ad to a user, rather it pays only if the user clicks on an ad. The broker selects which ads to show through an auction whereby advertisers bid against each other. In a PPC system, the broker maximizes revenue by ranking the competing ads according to the $Bid \times ClickProbability$ product, and transmitting the highest ranking ads to the client where they are displayed in rank order. Of course, the broker does not know the precise click probability for every ad. Rather, the broker tries to predict the click probability as best it can. This prediction is based on a number of inter-related factors such as the ad keywords, the landing page keywords, the user search terms or keywords associated with the web-page being browsed, stored user characteristics, and so on. For example, Microsoft incorporates at least seven and perhaps many more such factors in its Bing search advertising auctions.

The user profile has a strong effect on click probability. To give a simple example, say a user searches for "running shoes". Whether the user is a man or a woman, or prefers brand-name products or discount products, plays an

[1] Also called Cost per Click (CPC).

important role in which running shoes ad he or she is more likely to click on. In a private-by-design advertising system, the broker has no access to the user profile: if the auction takes place at the broker in the same way that it does today, then the user profile will not be factored into the result. Therefore the highest $Bid \times ClickProbability$ ads will not be selected, leading to less revenue than should otherwise be possible.

In this part of the thesis, we characterize the problem of running an auction that leverages the user profile while preventing the broker from reconstructing it, and propose three basic solutions. Taking a pragmatic approach to the problem, we look for a good trade-off between strict privacy guarantees and practical business and deployment concerns. As such, we explore the pros and cons of the three approaches in terms of not just privacy (both user and advertiser), but also revenue, overhead, and vulnerability to attack. We use around 2TB of auction traces from Microsoft Bing to guide and validate the design choices.

The remainder of this chapter describes how current online advertising systems such as Google and Microsoft work. It then outlines an abstract alternative advertising model that captures key aspects of the existing proposals for *private-by-design advertising*. Finally, it provides an overview of the related work. In the process, we establish terminology and define the basic components.

2.1 Current Second Price Ad Systems

In current ad systems [31, 46, 22], *advertisers* submit *ads* to a *broker*. Associated with each ad is a *bid*, one or more *keywords*, and optionally some targeting information like demographics (location, age, gender) or interests. When a *client*

computer does a search or receives a web page with *adboxes* (space to place an ad), the broker identifies the ads that match the search terms or keywords associated with the web page, and runs an auction. The auction ranks the selected ads in order of highest expected revenue ($Bid \times ClickProbability$), and transmits some number of ads to the client. As already discussed, many factors are considered in estimating click probability. We refer to all of these factors taken together as a *quality score* Q, where a higher value means higher expected click probability. Denoting bid as B, the ranking then is in order of the product ($B \times Q$).

When a user clicks on an ad, the ad ID is transmitted to the broker. The broker computes Cost per Click (CPC), that is, the price that the advertiser must pay, using a *generalised second-price auction* [25]. In this approach, the price paid is pegged to the bid of the ad that is ranked immediately below the clicked ad. To give a simple example, suppose that advertiser A is willing to pay as much as \$5 for a click, and advertiser B is willing to pay \$10. In a second-price auction, A could go ahead and bid \$5 and B could bid \$10. B would win, but would only pay the so-called second price, which is only incrementally more than A's bid, say \$5.01.

Second-price auctions allow bidders to bid the maximum that they are willing to pay, rather than frequently modify their bid in search of the value incrementally higher than the next lower bidder. Specifically, the CPC is computed as:

$$CPC = B_n \left(\frac{Q_n}{Q_c} \right)$$

where B_n and Q_n are the bid and quality score of the next lower ranked ad, and Q_c is the quality score of clicked ad. This CPC formula captures the minimum amount the advertiser would have had to bid to beat the next-ranked ad in a

first-price auction. Note that it prevents the advertiser from paying more than it bid, even when the next-ranked in fact bid more.

2.2 Abstract Private-by-Design Advertising

Although the private-by-design advertising systems cited in Section 1.2 differ in significant ways, they all share several key design components. All systems propose a *software agent* that runs at the client and generates a user profile. All systems at least share the privacy goal that this profile not be revealed. All these designs propose that the broker transmit multiple ads to the client, not all of which match the user profile. In this section, we describe an abstract private-by-design advertising system that captures key aspects of the existing proposals.

The principle components of the abstract private-by-design advertising system include those of today's tracking systems, the *broker*, the *client*, and the *advertiser*. A *user profile* is stored at the client (i.e., the user's computer, or a device trusted by the user: the distinction is not important for our purposes). Each ad is associated with targeting information. The user profile is defined as that information needed to determine how well an ad's targeting matches the user. To produce the user profile, the client monitors user behavior (i.e., the user's searching, browsing, purchases, and so on).

The privacy goals of the abstract private-by-design system are:

- **Anonymity**: the broker cannot associate any unit of learned information with any user personally identifiable information (including network address), and

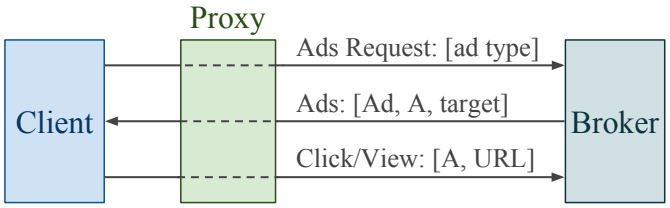


Figure 2.1: Abstract private-by-design advertising system. All communications between the client and the broker are relayed by an anonymizing proxy. [x] denotes encryption of x.

- **Unlinkability**: the broker cannot associate separate units of learned information with a single (anonymous) client. This prevents a broker from building up a user profile, and then associating it with a known user using externally gathered knowledge.

The broker is assumed to be honest-but-curious. We believe that this is close to reality (brokers like Google can generally be trusted to do what they claim they are doing). Nevertheless, we believe it is wise to avoid making it possible for brokers to obtain high-value information through hard-to-detect cheating, and our designs reflect this belief.

Figure 2.1 illustrates the basic architecture and message exchange of an abstract private-by-design advertising system. The network layer address of all messages is anonymized, which we represent as an anonymizing proxy. Messages are encrypted to prevent viewing by the anonymizing proxy. The client requests a set of ads of a given type (i.e. for a given product or service). The request must be generic enough that a substantial set of clients can have legitimately made the request (i.e. K-anonymity and L-diversity). A set of ads matching the type, each with identifier A and associated targeting information, are transmitted to the client and stored. When an adbox is presented to the client, for instance on a web page, the client selects among the stored ads those

that best match the user profile, and puts them in the ad box for viewing by the user. The client reports the view and click of the ad A on a webpage with the given URL. There is nothing in the messages that allows the broker to link different messages as coming from the same user.

This private-by-design model has two necessary channels of communications between the client and the broker, ad delivery, and view and click reporting. Both channels present opportunities for the broker to learn information contained in user profiles, and current private-by-design advertising systems protect these channels. Any new information that must be conveyed for the purpose of the auction must also be protected.

2.3 Related Work

Most of the work related to privacy preserving auctions revolves around cryptographic protocols designed to protect privacy of the submitted bids. Depending on the underlying security model these proposals can be classified into the following three categories. In the first category, there are protocols that rely on computation that is distributed among auctioneers who jointly determine the outcome of an auction using threshold multi-party computation (e.g., [37, 41, 55, 40]). The second category of protocols introduces a semi-trusted third party, aka an "auction issuer" or "auction authority", in addition to the auctioneer, and uses asymmetric multi-party computation technique, such as Yao's garbled circuit (e.g., [9, 12, 18, 44, 47]). Finally, protocols in the third category allow bidders to cooperatively compute the auction outcome without requiring on any trusted third party, instead they rely on the intractability of the

decisional Diffie-Hellman problem (e.g., [14, 15, 16]). The primary goal of all these proposals, and many others not cited, is to keep bids and selling price secret from the auctioneer and other auction participants. The problem that we address is different. We are primarily concerned with protecting the user, not the bidder (i.e., advertiser). Indeed the user does not exist in the prior work. In any event, the high computational and communication complexity imposed by aforementioned secure auction protocols make them impractical for our problem.

A large body of work in secure auctions [28, 45, 52, 48, 13, 51, 11] focuses on verifiable auction integrity developing a number of novel cryptographic techniques and distributed protocols. For example, VEX [11] proposes using hash-chains to improve integrity in the context of online ad exchange. Unlike the aforementioned work, our main target is ad networks, and our primary goal is to preserve user privacy.

An alternative approach to embedding privacy in online auctions relies on shifting the trust from an auctioneer to hardware, for example, to a hardware implemented secure co-processor [50]. In other words, the auctioneer is replaced by a combination of hardware and software that can be trusted by all the parties involved in the auction. The auction software (source and signed executable) is published and can be verified by the participants. Remote authentication is then used to ensure that the running software is not tampered with. We believe that such hardware-oriented approach is complementary to private auctions designs, and can be applied most effectively to implement a variant of Rank-at-Third-Party (Section 3.2.3).

CHAPTER 3

DESIGNING ONLINE AD AUCTIONS

In this chapter, we broadly explore the design space proposing three solutions for online ad auctions in private-by-design advertising, and then discuss the pros and cons of these solutions.

3.1 Auction Goals

The privacy goals for the auction component of a private-by-design advertising system are the same as described in Section 2.2: anonymity and unlinkability. This section describes the goals of the auction itself.

The primary goal of the auction in a private-by-design advertising system is to provide a second-price auction mechanism that achieves close-to-ideal ranking of ads (i.e., in order of $Bid \times ClickProbability$). For today's tracking advertising systems, leveraging the user profile is straightforward, since the broker itself accumulates and maintains this information. In a private-by-design system, the broker does not have user profile information, but does have other information that goes into the quality score Q. In other words, part of the information used to produce Q is in the broker, and part is in the client. Therefore, we define a *user score* U which directly reflects the effect of the user profile, when matched against an ad's targeting information, on click probability. We define a second quality score G, that reflects the remaining "global" information known to the broker.

Specifically, this results in an ideal ranking and CPC of:

$$Rank \Rightarrow B \times G \times U \tag{3.1}$$

$$CPC = B_n \left(\frac{G_n \times U_n}{G_c \times U_c} \right) \tag{3.2}$$

For example, U could be a positive real value greater or less than 1 that raises or lowers the click probability proportionally to its effect on the click probability defined by G. Section 3.3 briefly discusses how U may be computed. In particular, the bid B, quality score G, and user score U of an ad may change at any time. Ideally, the ranking used when displaying ads to users is based on current values of B, G, and U. To keep auction overhead low, however, it may be necessary to work with slightly out-of-date values for B, G, and U.

In current tracking advertising systems, the click normally takes place almost immediately after the view, and so CPC is normally computed shortly after the ranking. As a result, the parameters that go into determining the ranking (B and Q) do not change much between ranking and CPC. In private-by-design advertising systems, as explained later, some time may pass between when B is set by the advertiser and when the ad is ranked, or between when an ad is ranked and when CPC is calculated. Therefore, we set the following goals with respect to ranking and CPC calculation:

- The B, G, and U used for CPC calculation are the same as the B, G, and U used for ranking. Note in particular that if they are not the same, then it is possible for instance for the CPC to be higher than the submitted bid of the clicked ad.

- The delay between ranking and CPC calculation is small enough that the churn in B, G, or U does not have a significant impact on rankings, CPC values, and broker revenue.

What exactly comprises user score U depends on the client profiler and can vary from system to system. We can, however, classify user information into three time frames. At the time frame of months or even years are user demographics like gender, location, language, age, salary, and so on. User interests can also last years (e.g. coin collecting), but more typically last weeks (a new car), days (a new pair of shoes), or minutes (a pizza). If we assume that matching ads to the content of a web page or search page increases click probability, then user score can change in seconds or less. For instance, a user might be interested in tennis and music, but the user score for tennis ads may increase while the user is looking at a tennis website, and vice versa for music ads.

We do not make any assumptions about the relative importance of B, G, or U. An ideal auction design however must allow for this flexibility.

Besides the basic goal of running an auction that leverages the user profile while prohibiting the broker from reconstructing it, there are a few additional related goals that are important:

- to maintain the privacy offered to the advertisers themselves. In particular, to prevent advertisers from learning each others bids and budgets.
- to maintain the level of click-fraud defense in current tracking systems.
- to minimize the overhead of the auction.

As will become apparent in subsequent sections, our designs do not perfectly achieve all of these goals. Rather, our designs offer trade-offs between these goals.

Note finally that it is possible that the value of U may correlate with the probability that the user will buy the product or service being advertised assuming that the user has clicked. If this is the case, then the advertiser would want to express multiple bids as a function of U, since different U's would produce different revenues for the advertiser. We do not address this capability in this thesis, but rather leave it for future work should it turn out to be important.

3.2 Detailed Designs

To run the auction specified in the previous section, the system computing the ranking must have access to the bid B, the broker quality score G, and the user score U. This means that either 1) B and G are sent to the client, 2) U is sent to the broker, or 3) B, G, and U are all sent to a third party. These basic approaches are explored in the following sections.

3.2.1 Rank-at-Client (RaC)

In this approach (see Figure 3.1), the following information is transmitted with the ad to the client along with everything else required by the advertising system (e.g. targeting information, not shown):

A: The ad ID.

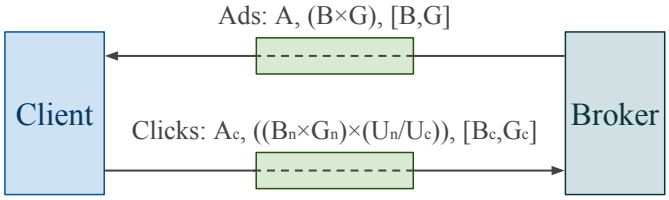


Figure 3.1: Rank-at-Client. A is an ad ID unique to the ad. The subscripts 'c' and 'n' refer to the clicked ad and the next ranked ad respectively. [x] denotes encryption of x. Messages between the client and the broker pass through an anonymizing proxy (dashed lines represent encrypted messages).

$(B \times G)$: A single value which is the product of $(B \times G)$.

$[B, G]$: The individual values B and G, encrypted with a symmetric key known only to the broker.

When a collection of ads arrive at the client, it ranks all ads using $(B \times G \times U)$. Note that in this case U is current, while $(B \times G)$ can be somewhat stale depending on the system delays. If the user clicks on an ad, then the client computes the following values and transmits them to the broker:

A_c: The ad ID of the clicked ad.

$((B_n \times G_n) \times (U_n/U_c))$: A single value which is the $(B_n \times G_n)$ product of the next-ranked ad times the ratio (U_n/U_c) of the user score of the next-ranked ad U_n and the user score of the clicked ad U_c.

$[B_c,G_c]$: The encrypted B and G for the clicked ad as received earlier from the broker.

Upon reception of this message, the broker decrypts $[B_c,G_c]$. It uses G_c and $((B_n \times G_n) \times (U_n/U_c))$ to compute the CPC as shown in Equation 3.2. The broker

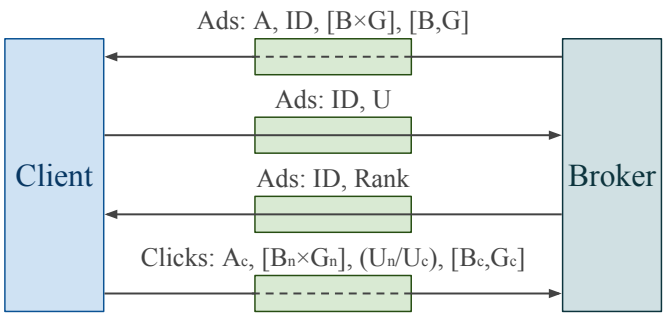


Figure 3.2: Rank-at-Broker. ID is an Ad ID unique to the combination of ad and client. Solid lines represent clear-text messages forwarded by the proxy.

also compares the resulting CPC with the decrypted B_c. If $CPC > B_c$, then the broker knows that the client is engaged in click fraud, and the broker can ignore the message. If $CPC \leq B_c$, then the broker can accept the message, although this does not mean that the client is not engaged in click-fraud. Other mechanisms, such as statistical analysis, must be used to detect it as is done today.

Variation: It may not be necessary to transmit the encrypted values [B_c,G_c]. This is because B_c and G_c can be looked-up using the ad ID A_c. The danger here is that the looked-up values may be different from the B_c and G_c values used to rank the ads. How different depends on the level of churn in B and G values, which we found to be minimal in the Bing auction trace (Chapter 5). Therefore, it may well suffice to use looked-up values rather than values stored along with the ad at the client. Note that this variation applies to RaB and RaT as well.

3.2.2 Rank-at-Broker (RaB)

One concern with RaC is that the value ($B \times G$) exposes information about the advertiser (see Section 4.1). This can be avoided if the ranking is done at the

broker as shown in Figure 3.2. The RaB scheme presented here is a substantially improved version of the approach first proposed in [33].

Along with the ad, the broker transmits the following to the client:

A: The ad ID.

ID: An identifier unique to this specific delivery of this ad (among all other deliveries). In other words, the same ad delivered to other clients would have a different values of ID.

$[B \times G]$: A single value which is the product of $(B \times G)$, encrypted with a symmetric key known only to the broker.

$[B, G]$: The values B and G, encrypted with a symmetric key known only to the broker.

The client computes a user score U for each ad (in the absence of knowledge of what web page the ad may be shown on). In order to obscure the user profile, the client assigns a random value for U for those ads for which the client has a very low user score (for instance because the demographic does not match that of the user).

Clients transmit the ID,U tuples to the broker via a proxy. These messages are not encrypted. The proxy also remembers which IDs were received from which clients.

For each received ID, the broker looks up the current values of B and G. It then uses B, G, and U to rank each received ad among a large number of recently received ads (say, those received over the last hour), and associates a

rank number *Rank* to each ad. Closely ranked ads may have the same ranking number. The broker transmits the ID,*Rank* tuple back to the proxy.

The proxy looks up which client is associated with each ID, and forwards the message on to the client. The client disregards the ads related to the low user scores, and uses the remaining ranking for selecting ads to put in ad boxes.

When a user clicks on an ad, it transmits the following information to the broker:

A_c: The ad ID of the clicked ad.

$[B_n \times G_n]$: The encrypted $(B \times G)$ for the next-ranked ad received earlier.

(U_n/U_c): A single value which is the ratio of the user score of the next-ranked ad U_n and the user score of the clicked ad U_c.

$[B_c,G_c]$: The encrypted B and G for the clicked ad received earlier.

Upon reception of this message, the broker decrypts $[B_n \times G_n]$ and $[B_c,G_c]$. It uses G_c, $(B_n \times G_n)$, and (U_n/U_c)) to compute the CPC as shown in Equation 3.2. As with RaC, the broker also compares the resulting CPC with the decrypted B_c for click fraud.

The proxy prevents the broker from learning the identity of the client whose ads are being ranked. The per-client-per-ad unique ID prevents the proxy, which does know the client identity (network address), from learning which ad, and therefore what targeting information, is being referred to.

Even with the noise added to low user scores, we are concerned that the non-noise user scores can be interpreted at the broker as a kind of fingerprint over

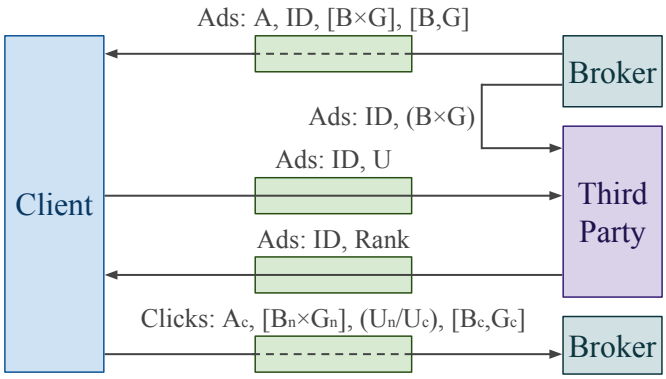


Figure 3.3: Rank-at-Third-Party

the set of ads (i.e., ads targeted to men should have uniformly higher scores for men, and lower scores for women). In this way, the broker could potentially tease out the profile of users.

3.2.3 Rank-at-Third-Party (RaT)

This approach (see Figure 3.3) is similar to RaB, but prevents the fingerprinting mentioned above. The main difference between RaT and RaB is that in RaT the broker additionally sends the unique ad IDs and (B×G) products to a third party system which is trusted not to collude with the broker. This information must be delayed long enough that the third party system cannot use a timing attack to correlate the values associated with a single user. This third party also receives the user scores from the clients, and based on this information, ranks ads in the same way the broker does in the RaB approach. Since, unlike the broker, it does not know which ads were transmitted to the same client, it cannot fingerprint the clients.

3.2.4 Homomorphic Encryption Variant (RaC, RaB, and RaT)

A variation on all three auction designs is to use homomorphic encryption (e.g., ElGamal [26]), which allows for multiplication operations on encrypted data. This may be used to defend against certain attacks by the broker as described in Sections 4.1 and 4.3. When a user clicks on an ad, the client encrypts (U_n/U_c) with the broker's public key. In the case of RaC, it also encrypts $(B_n \times G_n)$ with the broker's public key. In the case of RaB and RaT, the broker provides the encrypted $[B_n \times G_n]$, but using its public key instead of a symmetric key. For all three schemes, the broker provides $[1/G_c]$, again encrypted with the broker's public key. Using homomorphic property of the encryption, the client is able to calculate:

$$[B_n \times G_n] \times \left[\frac{U_n}{U_c}\right] \times [1/G_c] = \left[B_n \left(\frac{G_n \times U_n}{G_c \times U_c}\right)\right]$$

and transmit the resulting value in the click report. Upon receiving a click report, broker decrypts the value to obtain the CPC. Although homomorphic encryption is relatively expensive, there is no need to do the operation in real-time. Rather, the client can do the operation when it has spare CPU cycles before transmitting, and the broker can likewise run the operations later on as batch processing.

3.3 Computing User Score

So far, we have assumed the existence of a user score U that, when multiplied with the quality score G produces the expected click probability at the client for a given ad. Because clicks are relatively rare, it may be difficult to estimate U at the client based purely on the click history of the client. Therefore, we re-

quire that the broker anonymously and unlinkably gathers detailed click statistics from clients in order to improve click probability estimates at individual clients. In what follows, we sketch out an approach.

There are a number of measurable attributes $X = \{x_1, x_2, ...x_L\}$ at the client that may help in prediction of click probability. For instance, the level of interest (high or low) in the ad's product or service, the quality of the match between the targeting and the user, the context of the webpage, as well as the user's historic CTR. The idea is that each client reports this information anonymously to the broker for each ad that it views and clicks. These reports contain: {Ad-ID, X, click}, where X is the values of the attributes, and 'click' indicates whether or not the ad was clicked. Given this information from many clients, the broker can determine the effect of the attributes on click probability, and convey this information to the client as a function f of the attributes such that $U = f(X)$, along with the ad. This allows the client to compute U by measuring the attributes and plugging them into the provided function. As mentioned, U in RaC can be computed at viewing time with the latest set of attributes without churn issues since that is when the ranking takes place. The function f for a new ad can be initially set to that of similar existing ads until enough data for the new ad is gathered. The details of this are left as future work.

One concern is that the set of attribute values X is unique for a given user. Several factors can mitigate this concern. First, the attributes may be fairly coarse-grained, thus broadening the set of users to which they apply. Second, some of the attributes may be hard to correlate using external knowledge, such as the user's CTR. Third, attributes like level of interest change from interest to interest, and even within an interest over time, and therefore are hard to link to

the same user. Fourth, some attributes are not specific to the user, for instance webpage context. Finally, the only information beyond the attribute values that is leaked is the ad viewed. In particular, the user's click-stream is not exposed. We believe that it is reasonable to establish public policies that determine the nature of the attributes in such a way that meaningful privacy is preserved.

CHAPTER 4

AUCTION ANALYSIS

This chapter analyzes the three types of auctions in terms of privacy, auction quality, and potential attacks on the auction component.

4.1 Privacy Properties

In this section, we look at the information that is conveyed for the sake of the auction between honest-but-curious players, and determine whether it constitutes a privacy threat. In Section 4.3 we relax the assumption of honest-but-curious players.

Broker analyzes (U_n/U_c) (RaB and RaT)

In order to exploit this value to gather more information about the user profile, the broker would have to first tease apart the values of U_n and U_c, then use the value combined with the ad targeting to reverse engineer the user profile, and then use the user profile knowledge to link together multiple reports. The first step may be made difficult by making (U_n/U_c) relatively coarse-grained, thus making it harder to uniquely factor out its components. The second step is made difficult simply by the sheer number of clients that are likely to have similar user scores. Thus, we conclude that exposure of (U_n/U_c) does not constitute a serious threat.

Broker analyzes $((B_n \times G_n) \times (U_n/U_c))$ **(RaC)**

This value is more difficult to reverse engineer than (U_n/U_c), and is therefore also not a threat.

Client analyzes $(B \times G)$ **(RaC)**

An advertiser can use this value to determine the broker quality score G assigned by the broker to its own ads. This can be done by the advertiser simply creating a client that receives its own ads, and using the known value of B to factor out G. Whether this is a problem needs to be decided by the broker, though we point out that today Google reveals a coarse-grained quality score to its advertising customers.

Transmitting the product $(B \times G)$ to the client also reveals the overall ranking of an ad to anyone running a client, including the advertiser's competitors. From this, they can also roughly estimate the advertiser's bids. It is not clear that this is a problem, for two reasons. First, in today's advertising systems, an advertiser can see how its competitors rank relative to itself simply by observing how ads are displayed. RaC makes it easier and cheaper to obtain this ranking information, but does not fundamentally change an advertiser's ability to do so. Second, historically in traditional advertising (print, TV, radio), advertisers can monitor how much advertising their competitors do, and can generally know the cost of that advertising. While certainly all things being equal advertisers would like to keep this information secret, historically the inability to do so has not, for the most part, prevented companies from advertising.

If exposing the product ($B \times G$) to the client is an acceptable privacy loss, then RaC should be the preferred auction method for its overall simplicity and lower overhead. If it is not acceptable, then RaB and RaT, which both avoid exposing this information, may be preferred.

4.2 Auction Properties

In this section, we discuss the various shortcomings of each of the approaches with respect to the auction properties, especially ranking results and revenue.

4.2.1 System delays

There are several potential delays in the private-by-design advertising systems that can change both the rankings and the computed CPC. With RaC, there is a delay between when the ad was transmitted and adbox time when the ranking takes place. With RaB and RaT, there is a delay between when the ranking occurs and adbox time when the ranking is actually used. In either case, the bid B or the broker quality score G used for ranking may no longer be correct, and an out-of-date ranking takes place.

During the design of the auction approaches, these delays were a major concern. As it turns out, at least for the auction data from Bing search advertising auctions (Chapter 5), the delays have only a minor impact on both broker revenue and advertiser costs, even when the delay is several hours or a day. Nevertheless, this may not be the case for other systems or future systems, and so it remains important that these delays are engineered to be minimal. This

could be done, for instance, by having clients frequently request small numbers of new ads.

4.2.2 Client selection

A problem encountered by the private-by-design advertising is that the broker does not know which clients are best to send an ad to. For instance, suppose that some number of clients M have requests ads for watches. The broker does not know which clients may be interested in cheap watches, and which in expensive. The advertiser, however, might not have enough budget to pay for all the clicks that would result if all watch ads are sent to all interested users.

Lets assume that the broker knows the clicks per delivered-ad rate. From this, it can determine the number of clients N that should receive the ad without exhausting the advertisers budget. If it randomly chooses N clients among the interested clients, then it will not be sending all ads to the most interested clients.

One way to solve this problem is for the broker to go ahead and send the ad to all interested clients, but to also send a parameter giving the minimum user score that a client must have in order to show the ad. This way, only the best matching clients will show the ad. The broker may be able to establish the expected click per delivered-ad rate for various user scores, and therefore predict the setting of the user score based on the number of clients. If the broker predicts too high, then it can lower the minimum user score and send this to clients, thus causing more clients to show the ad.

4.2.3 Overhead and Latency

RaB and RaT add load to real-time advertising systems (systems that do not prefetch ads at the client), to the point where neither is a very attractive option.The extra round trip to the broker (RaB) or third-party (RaT) has to happen in real-time because the ads must be delivered in time to render the web page. This is compounded by the fact that a number of ads must be shipped around for each page load.

If all the ads are to be delivered to the client, the sheer volume of ads that must be transmitted to the broker or third-party for ranking can also be a scaling challenge. The Bing trace contained 15M unique ads for a single day (see Section 5.3), with an average ad lifetime of roughly 9 days. This translates into a little over 1.7M new ads per day per client. Each of these ads is given a user score by each client, which is then transmitted to a broker or third party for ranking. The Bing trace also counted 14M unique clients. This then translates into 24 tera-ads per day that must be ranked. Fortunately, this ranking function can be split over many machines (with each taking some fraction of the total number of ads to rank) without hurting the accuracy of the ranking significantly (assuming that each machine has a representative sample of ads). Therefore, while challenging, this sort is doable.

Though this is not related to the auction per se, note that each ad is roughly 250 bytes of text including the URL. Even ignoring client updates to B and G values over the lifetime of an ad, this still requires 425MB of ads downloaded per day per client (uncompressed), or about 43MB compressed (in bulk).

4.2.4 Auction Scope

An important aspect of the auction is the scope of the auction, by which we mean the set of ads that compete in any given auction. As a general rule, the more ads that compete, the higher the CPC. This is simply because the more ads there are, the more probabilistically likely the next-ranked ad will have a $(B \times G \times U)$ closer to that of the clicked ad. On the other hand, the larger the auction scope, the less fair it is in the sense that very different types of ads must compete. A local pizza store may not wish to compete with Mercedes for ad boxes.

The auction scope for search or contextual systems like Bing and Google is the set of ads whose keywords match that of the search or web page. Today's ad exchanges, where advertisers bid in real time, typically for adboxes on premium publishers, have a potentially much broader scope because any advertiser can bid. The auction scope in a private-by-design advertising system is tunable. It may be all ads in a client, or all ads within an ad type (i.e. an interest). What is more, interests may be hierarchical (sports/tennis/clothing/shoes), and may be more general or more specific, thus allowing for substantial flexibility in auction scope.

4.3 Attacks

In this section, we relax our assumption of honest-but-curious players, and consider a number of malicious attacks and defenses.

Client click fraud

The client can commit a form of click fraud by lying about the value of $((B_n \times G_n) \times (U_n/U_c))$ (RaC) or (U_n/U_c) (RaB or RaT). By inflating or deflating these values, the client can cause advertisers to pay more or less, and cause publishers to earn more or less. At a high level, this is very similar to normal forms of click fraud that occur today, and in this sense our auctions do not allow fundamentally new forms of click fraud. Privad describes how to defend against click-fraud even with anonymizing brokers [32]. The same method may be used here. The basic idea is that the proxy tags reports with a per-report unique identifier. If the broker suspects click fraud, it informs the proxy of the report ID of the suspicious report. If a given client is suspected more times than some threshold, its reports can be tagged by the proxy as coming from a suspected client. In some cases the broker may suspect click fraud simply because the second price is impossibly high (i.e., higher than the first price bid). In most cases, however, the broker may use a variety of additional mechanism to detect an ongoing click fraud attack. As described in Section 6.1, these mechanisms range from using statistical analysis of historical per-publisher and per-advertiser click-trough rates to proactively setting up "bait ads".

Proxy fingerprints client user scores and resulting ranking (RaB and RaT)

It is difficult but conceivable in RaB and RaT that the proxy could determine user profiles through observation of the client user scores and rankings. For instance, the proxy could establish a number of fake clients that pretend to have various profile attributes, and establish fingerprints of the resulting user scores and rankings. One way to do this might be to determine $(B \times G)$ given user

scores and corresponding ad ranks, and use these values as the fingerprint. The proxy could then compare these fingerprints with the corresponding fingerprints of real clients. It could be that the signal-to-noise ratio is high enough to successfully pull off this attack. One way to prevent this would be to encrypt user scores and rankings. The user scores could be encrypted using the brokers (RaB) or third-party's (RaT) public key, and the rankings could be encrypted using symmetric keys created by the clients and conveyed securely to the broker or third-party. These symmetric keys would be frequently modified to prevent the broker or third-party from linking user scores with the same client, and possibly launching a similar fingerprint attack.

Broker manipulates $[B_n \times G_n]$ (RaB, RaT)

A malicious broker could launch an attack on a private-by-design advertising system to identify clients by inserting unique IDs into the encrypted fields $[B, G]$ or $[B \times G]$. Once a client is identified in this way, unlinkability is lost, and the broker can build up client profiles. The broker can then potentially identify the client through external means. There is some cost to this approach, as the broker must "waste" ads to do the tracking[1]. The homomorphic encryption variant described in Section 3.2.4 defends against this attack. Because the client multiplies the received encrypted fields with other fields, the values generated by the broker are obscured.

[1] One might argue that the same attack can be launched simply be creating unique Ad IDs transmitted in the clear. However, this attack can at least be detected by third parties, for instance running honey-farms of clients.

Broker sees client user scores (RaB)

The noise added to user scores, combined with the anonymization of the client address and unlinkability of interest channels makes it extremely difficult for the broker to build up a user profile. Having said that, we do not prove that it is impossible, and it could well be that a clever broker could figure out how to create a kind of user-profile fingerprint on otherwise anonymized channels. From this, the broker could in theory link together channels with identical fingerprints, thus violating unlinkability claim. The RaT approach eliminates this possibility altogether.

4.4 Discussion

Overall, we find that RaB is a weak scheme because it opens up a fingerprinting attack at the broker. RaT solves this problem, though at the expense of requiring yet another administratively distinct and non-colluding entity. Nevertheless, we consider it to be a better alternative to RaB.

If exposing the $(B \times G)$ product to the client is not a problem for the broker and advertisers, then RaC is better than RaT because it is simpler, incurs less overhead and latency, and does not require the third party. In addition, it has no issues here with respect to user score churn, because ranking takes place at ad view time. If exposing $(B \times G)$ is a problem (this information, however, is even today indirectly revealed through the ad positions in adboxes), then RaT appears to be a reasonable approach.

CHAPTER 5

TRACE-BASED SIMULATION: EFFECT OF CHURN

Section 4.2 describes how various delays in all three auction systems may distort rankings and CPC computation. How detrimental this delay is depends on how much churn there is. Churn may affect rank, CPC value, and ultimately revenue. RaC is affected by churn in B and G, while RaB and RaT are additionally affected by churn in U. In this chapter, we use trace data from Microsoft's Bing advertising platform to study in depth the effect of auction delays on B and G from both the advertiser and broker perspective. We find that, while churn exists, it has only a negligible impact on broker revenue and advertiser costs.

5.1 What Causes Churn?

$B \times G$ for an ad changes when either B or G changes. B can change in one of three ways: first, the advertiser can manually update the bid; second, the ad network can automatically update the bid (as directed by the advertiser); third, a third-party may update the bid on the advertiser's behalf. Each of these has different churn characteristics:

Advertiser: Manual updates, we believe, cause very little churn since they are reactive over a long feedback cycle. Advertisers receive updated campaign information (i.e., how many clicks, actual amount charged, budget left) at fairly coarse intervals (few times a day). This limits the number of informed changes to their campaigns.

Ad Network: The advertiser can invoke functionality provided by the ad network to optimize his bidding strategy. For example, the ad network may al-

low the advertiser to set a preferred rank (e.g., position 4), and the ad network automatically lowers or raises the bid to satisfy the request based on the market. Other examples may include automatically modifying bids to meet a target number of impressions per day (while still being charged only for clicks), or modifying bids based on time of day etc. Some of this functionality (e.g., modify bids based on time-of-day) can be implemented in the client and would therefore not result in any added churn. Other functionality (e.g. preferred rank) tends to be implemented today as a periodic update (once every few hours).

Third Party: Search Engine Optimization (SEO) companies optimize their client's bidding strategy in real-time [21] e.g., based on trending terms, real-time click-through rates, etc. This could potentially result in high bid churn, however, due to the premium nature of these SEO services, only a small number of ads would be affected.

Aside from changes in B, G can also change. Recall G in our model is a function of what the broker knows: G is computed based on the ad (past CTR, landing page quality, etc.). G is largely a property of the ad itself, which we do not expect to change quickly or dramatically. In any event, our Bing auction trace unfortunately does not allow us to validate our assumption since it does not isolate user-derived components of G from other components.

5.2 How Does Churn Affect Auctions?

Today auctions take place at the time when an ad is displayed to the user; ranking and CPC calculations can immediately reflect any changes in B or G. Privacy compatible auction designs described in Chapter 3 are limited in terms of how

fast new B and G information can be incorporated. Since G does not rapidly change over time or can be engineered to remain relatively stable (e.g., using U to reflect short-term changes in click probability), the main source of churn is the changes in B. To understand the effects of churn in B values, we simulate auctions that use stale B information for ranking and CPC computation, and then compare the resulting ranking and CPC computation with auctions that use up-to-date B information.

5.3 Dataset

For our trace driven simulations, we sampled around 2TB of log data from Bing's auction engine spanning a 48 hour period starting September 1, 2010. The data covers over 150M auctions for over 18M unique ads shown to North American Bing search users across all search topics. The trace record for an auction lists all the ads that participated in it (whether the ad was ultimately shown or not), the bids corresponding to each ad, the corresponding quality scores, and which if any of the ads were ultimately clicked by the user.

5.4 Methodology

We re-compute auction rankings and the CPC for each auction in our dataset using stale bid information; we vary staleness from 1 minute to 2 days.

Auction rankings are re-computed using bid and quality data from the trace. Since our trace does not show when the advertiser updated the bid, we infer the

time based on multiple auctions that a given ad participates in. If the bid for the ad is the same for two consecutive auctions, we infer that the bid did not change during that interval. If the bid is different, we infer that the bid changed sometime between the two auctions; we use the mid-point as the time of change. To simulate an auction at time T with stale information from d minutes ago, we simply use the bids current as of time $T - d$ in our trace. The quality score in the trace is based on user features (e.g., search query), which correspond to U in private auctions; since the client always has the current value of U we use the same quality score for simulated auctions as in the trace.

CPCs are re-computed based on the re-computed auction rankings (using the second-price formula of Equation 3.2). In other words, for an adbox at time T in the trace, we compute the ranking based on bid values recorded at time $T - d$ and populate the adbox using resulting ranking. If the user clicked on an ad in this adbox, the bid of the next lower ranked ad B_n that we use in the CPC computation is the stale B_n taken at time $T - d$.

One limitation we face is that we cannot predict the change in user behavior when auction rankings change. Consider, for example, two ads A_1 and A_2 where in the trace they are ranked 1 and 2, while in the simulated stale auction they are ranked 2 and 1 respectively. If the user clicked A_2 in the trace, what might we expect the user to click in our simulation? One option is to model the user as clicking the same ad he clicked in the trace; thus in this case the user clicks A_2 in the simulation. Another option is to model the user as clicking the same position he did in the trace; in this case the user clicks position 2 (A_1 in the simulation). In reality, the user model is neither of these two extremes — it is well-known that both ad content and rank effect click-through rates (CTR) [27]. To account

for this, we simulate 5 user models: 1) same position, 2) 75% same position and 25% same ad, 3) 50%-50%, 4) 25% same position and 75% same ad, and 5) same ad. Thus we establish an envelope of possible user behavior to get a sense of the upper- and lower- bounds of our simulation results. Note that always clicking on the same ad is a strictly conservative estimate. This is because an ad that was clicked in the trace but is not shown to the user in our simulation (due to being ranked too low) would not get clicked; at the same time, under the same-ad model, an ad that was not shown in the trace (due to being ranked too low) and was therefore not clicked would have no chance of getting clicked even if it were to be shown to the user in the simulation. This asymmetry biases the simulation towards fewer clicks (and therefore lower revenues). The only user model immune to this limitation is the same-position model.

A second limitation we face is that we cannot predict how advertisers would change their bidding strategy in response to auctions being based on stale information. Enterprising advertisers or SEOs, may for instance, attempt to predict what bid they might want to make one hour hence, and enter it into the system well in advance. Advance bidding would reduce the effective staleness of information. For our purposes, we assume the bidding strategy does not change.

5.5 Simulation Results

Overall our simulations show that there is no appreciable change in broker revenue for using stale bid information; even in the most conservative cases, the revenue is within ±0.1% of today. For advertisers, while stale bids affect their auction rankings, they do so in a balanced manner with cases of higher-than-

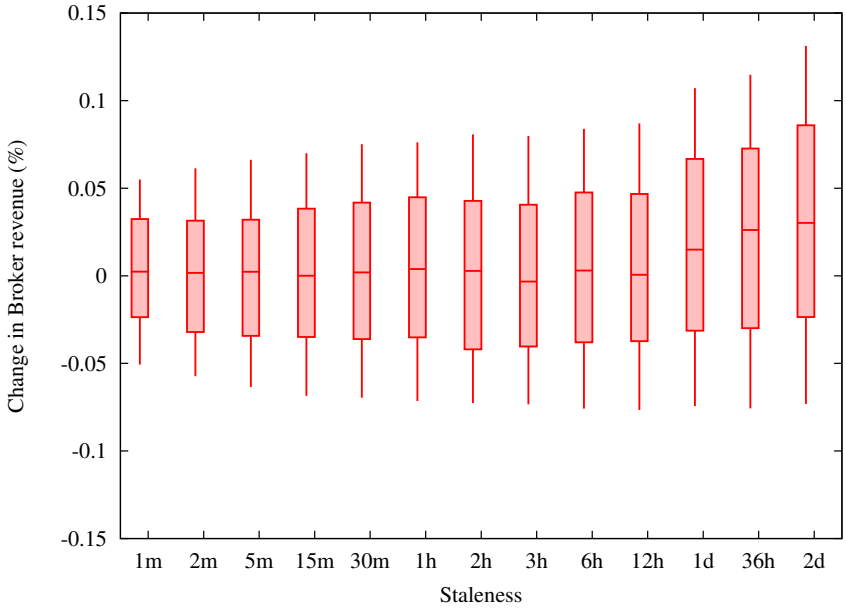


Figure 5.1: Change in broker's revenue

today rank canceling out cases of lower-than-today rank resulting in zero net change.

Figure 5.1 plots the change in broker revenue compared to today as a function of the staleness of information used and the user model. The x-axis varies the stateless of bids from 1 minute to 2 days. The box-and-whisker plot varies the user model with the top whisker showing the outcome where the user clicks the same position, and the bottom whisker showing when the user clicks the same ad; the top edge of the box shows 75% same position and 25% same ad, and vice versa for the bottom edge of the box; the line in the middle shows the 50%-50% case.

The first observation we make from Figure 5.1 is that under a 50-50 user model, change in revenue is practically 0% even with bid information as stale as up to 12 hours. Under the 75-25 and 25-75 models, the change is almost

always between ±0.05%, and only in the extreme cases 100-0 or 0-100 does it pass ±0.1%. More importantly, the change increases very gradually. This is good news since it means a private advertising system would not have a hard delay deadline beyond which there would be disproportionate change in revenue. Instead the system can strive to do the best it can, and reduce revenue change proportionally. The extremely gradual rate of change also means that system design trade-offs can be biased towards scalability and other engineering goals without much concern to revenue since it changes very little in the first place.

At first blush the effect of the "same-ad" user-model appears to be to reduce the revenue, but this is deceptive. As mentioned earlier, the more the user clicks on the same-ad (going from 0% to 100% from the top whisker to bottom), the more biased the simulation is towards fewer clicks and therefore less revenue. Recall that only the top-whisker is unaffected by this simulation bias.

The second observation we make from Figure 5.1 is the slight upwards trend of the top-whisker signifying higher revenues as more stale information is used. This suggests a consistent trend of advertisers (as a whole) reducing their bids over time. We do not know the cause of this trend.

Next we turn to the advertiser perspective. We compute for each ad the fraction of auctions where the user-visible simulated ranking increased or decreased compared to the trace, and whether the ad became visible or invisible due to being ranked high-enough or too-low as compared to the trace. Figure 5.2 plots the average of these numbers across all ads as a function of the staleness of bid information used.

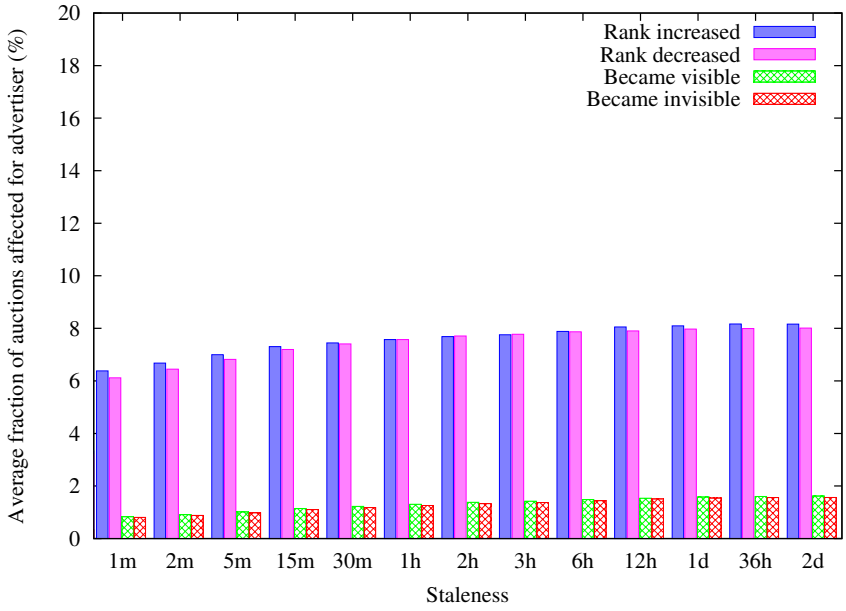


Figure 5.2: Fraction of auctions with modified rankings

We first observe that both increased ranks and decreased ranks are roughly equal, and so average to nearly zero. The same is true for ads becoming visible or invisible. While this is consistent with the revenue change in Figure 5.1 averaging out to zero, we note that there are other ways the revenue could average out to zero while being unfair to advertisers. For instance, fewer increased ranks could have been compensated by more cases where the ad became visible thus still resulting in zero revenue change while being unfair to the advertiser; luckily, this is not the case.

We observe next that there is a very small impact of staleness on change in ranks; it begins with around 12% of auctions for 1 minute stale data, and quickly converges to around 16%. The reason this number is high is because of the cascade effect — if a single ad jumps from a low rank to a high rank, it causes all the ads in between to register a "change" in rank; thus a single change in bid can affect up to ten ads. The impact, however, is very little; the ad jumping from

low to high might register a change of 10 ranks, however, the other 10 ads would register a change of only 1 rank each (not captured in the graph). Overall we found a median *net change* of 1 rank for every 820 auctions the ad participates in.

To summarize, based on extensive simulations across varying degrees of staleness and different user-models, there is little impact on broker revenue as compared to today, and little impact on advertiser fairness as compared to today.

Part II

Private-by-Design Advertising and Analytics Meet the Real World

CHAPTER 6

BACKGROUND AND RELATED WORK

As described in Section 1.2, several research projects have proposed an alternative "private-by-design" advertising model in an attempt to reconcile behavioral targeting and user privacy. Each of the proposed systems claims to be practical in that they provide both good privacy and high utility at reasonable cost. In this part of the thesis, we attempt to answer one simple question: "Is private-by-design advertising really practical?". To do so, we built, deployed, and evaluated a fully functional prototype of a private-by-design ad system based on the Privad design. Our deployment delivered functional ads in the sense that the ads were targeted to user interests, displayed on publisher webpages, linked to real shopping websites, and in fact led to actual purchases. Side-by-side with Privad, we also deployed a distributed differentially-private user analytics system, PDDP [19], that served as our primary means of gathering experimental data.

By bundling our system with a popular Firefox addon, we deployed it to over 13K opted-in users. Over a period of two months, the system was used daily by over 4800 active users on average, with more than 2000 users online at peak. In October 2013 alone, our backend received 1.1M ad requests and generated 9.5M ads. During that time, we registered 790K ads views, 417 ad clicks, and 4 product purchases. While minuscule by commercial standards, our deployment was big enough to allow us to preliminarily answer a number of important questions.

The lessons learned from this experiment contain both good news and bad news. Perhaps the most surprising of the good news is that our system pro-

duced click-through rates (CTR) on par with Google display ads. This is especially impressive given that ad generation in our system was fully automated, in contrast to Google where ads are designed by hand and fine-tuned over time.

Among the bad news, our experience suggests that differential privacy was a poor model for understanding actual privacy loss in our experiment. Based on the relatively small number of queries we made to our system (159 distinct queries generating 790K answers from 9395 unique clients), differential privacy's worst-case stance would suggest that a *substantial* proportion of our user base could have experienced privacy loss. In reality, no individual user information whatsoever was leaked through PDDP. Moreover, even if we had generated malicious queries, at best we could have learned one or two things about one or two users (assuming we had auxiliary information). This large discrepancy between the differential private model of privacy loss and actual privacy loss needs to be addressed by the privacy research community.

The high-level take-away, however, is that the system does appear practical. We could deliver effective targeted ads, obtain the information needed to pay publishers and bill advertisers, and gather statistical data giving us visibility into system operation and user behavior. The main limitation in our experiment is that we did not implement auctions or click-fraud defense. However, integrating with existing auction systems or ad exchanges does not appear feasible in the context of a small-scale academic research experiment.

In the rest of this chapter, we give a broad overview of the Privad architecture underpinning our prototype implementation and discuss the privacy guarantees provided by the system. We then outline the design of the PDDP system that leverages the same fundamental architecture to enable differentially private

data collection in distributed settings. Finally, we describe previous attempts at building and deploying a private-by-design advertising system.

6.1 Privad Model

A number of components in the Privad architecture play the same role as they do in today's ad deployments. These include *users*, *publishers*, *advertisers*, and a *broker* (ad network): users browse publisher webpages, and advertisers provide ads to brokers for display on those webpages. Privad defines two new components, the *client* and the *dealer*, and substantially modifies the role of the broker: user profiling and ad serving are delegated to the client software, which runs on the user's device, rather than in the cloud (i.e., at the broker) as it is done in today's deployments. The client monitors user behavior (i.e., the user's searching, browsing, purchases, and so on) and over time builds up a user profile. It then uses this behavioral profile to privately fetch ads from the broker and locally decide which ads should be presented to the user. Finally, the dealer is placed in between the clients and the broker to anonymously relay all communication and also help downscale click-fraud attacks.

The fundamental design principle in Privad is that private information about each user is kept on that user's computer, not in the cloud [34]. In a sense, users are still tracked. However, the tracking is done by a software agent running on the user's machine, and the information it gathers (the user profile) never leaves the user's machine. The challenge is to utilize the user profile to deliver targeted ad content while revealing the minimum amount of information from the user profile. Concretely, the privacy goals of the Privad system are formulated as follows [32]:

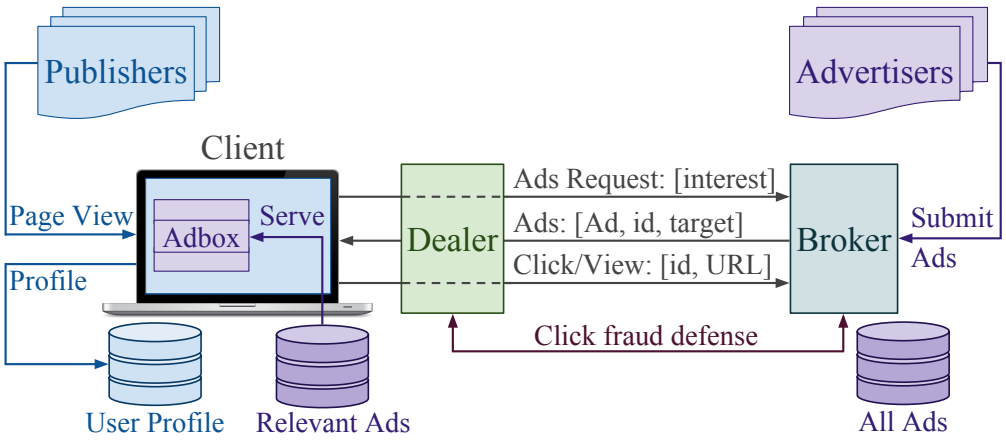


Figure 6.1: The Privad architecture. [x] denotes encryption of x.

- Anonymity: the broker cannot associate any unit of learned information with any user Personally Identifiable Information (including network address), and
- Unlinkability: the broker cannot associate separate units of learned information with a single (anonymous) client. This prevents the broker from building up a user profile, and then associating it with a known user using externally gathered knowledge.

Privad Operation

Figure 6.1 illustrates the basic Privad architecture and message exchanges between principal components. The client and the broker are separated by a proxy-like dealer, which strips away the network layer address of all clients' messages. The dealer can also coordinate with the broker to identify and discount fraudulent clicks as well as block clients suspected of click-fraud. The broker in Privad is assumed to be honest-but-curious. While this may be close to reality (brokers like Google can generally be trusted to do what they claim they

are doing), the system design nevertheless strives to prevent the broker from obtaining high-value information through simple but hard-to-detect cheating.

All message exchanges in Privad follow the same basic protocol: the client's request is encrypted with the broker's public key and contains a one-off symmetric key generated by the client, which is later used to encrypt the broker's response (e.g., the stream of ads sent to a client). Since the encryption is opaque for the dealer, it blindly forwards the messages without learning anything about the clients. As long as the broker and the dealer do not collude the system can offer privacy guarantees: the dealer prevents the broker from learning the client's identity or from linking separate messages from the same client.

User profiling software runs at the client. This software monitors the user's activity (e.g., search terms, browsing behavior, purchases made) and uses collected information to infer both the *interests* and *demographics* of the user. *Interests* reflect largely short-term user attributes (for instance, interest in products or services like sports.equipment.tennis or outdoor.lawn-care). *Demographics* incorporate long-term attributes such as gender, age, salary, and location. When the profiling software identifies a user interest, it anonymously requests a set of ads for the given interest category *type* (i.e., ads for products or services matching the user interest). The request must be generic enough that a substantial set of clients can have legitimately made the request (as described later, this is done by using a pre-defined interests categories). A set of ads matching the user request, each with an identifier *id* and associated *targeting* information, are transmitted to the client. The client software then filters out ads that do not match the profile and locally stores the rest.

One of the main requirements for an ad system is good targeting. The design of the Privad system reflects the view that an endhost client can achieve deep insight into a particular user as opposed to the shallow view into the aggregate behavior of a vast number of users available to the current ad networks. Thus, relatively simple techniques can be leveraged towards identifying the users interests and demographics. These techniques include monitoring the user's shopping activity, scraping the profile information on social networking sites, and observing what applications the user runs, what music the user listens to, and what websites the user browses.

Another key requirement for a private-by-design system is an efficient and privacy-preserving ad distribution channel. In Privad, advertisers upload their ads to the broker together with the targeting parameters and bid information. The broker distributes submitted ads to a fraction of clients through the dealer via a pub-sub mechanism. To receive ads, clients anonymously subscribe to a broad interest category combined with a few broad non-sensitive demographics (gender, language, region). The broker then periodically transmits new ads to relevant subscription groups. The dealer ensures anonymity of the message exchanges between the clients and the broker, while encryption makes the messages opaque to the dealer, so that it does not learn their contents. If the client detects multiple user interests, it issues a separate subscription for each interest, and the broker is unable to link the separate subscriptions to the same user. Note, however, that this distribution mechanism does not take into account the full set of interests and demographics of the user. As a result, the set of ads delivered to a client covers all demographics and fine-grained interests within a broad category. In other words, the client receives both ads that are and are not targeted to the user and has to locally filter out non-matching ads.

Privad relies on ad auctions to determine both which ads are shown to the user and in what order. As described in Chapter 3, in addition to bid information, ranking is based on both user and global metrics. Among other factors, user metrics incorporate how well the targeting information matches the user profile, and the levels of interest in similar ads in the past. Global metrics include advertiser's historical click-through-rates, the quality of the landing page, etc.

When an adbox is presented to the client, for instance on a webpage, the client evaluates the stored ads, selecting those that best match the user profile, and inserts them into the adbox displayed to the user. A report of this *view* (for each ad in the adbox) 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. These reports supply critical information to the broker required to bill advertisers and pay publishers. Additionally, the broker also uses these reports to provide advertisers with feedback on the effectiveness of their ad campaigns.

Privad recognizes the threat of click-fraud attacks that can be launched by unscrupulous users or clients against publishers, advertisers, or brokers. This threat is addressed in the system design, which requires that both the broker and dealer are involved in detecting and mitigating click-fraud. The mitigation strategy is for the dealer to identify fraudulent clients and suppress their reports. Privad proposes two mechanisms for identifying attacking clients. First, the dealer may flag an attacking client directly when the client transmits too many reports or subscription requests. Second, the broker can identify which publishers or advertisers are under attack, and indicate to the dealer which re-

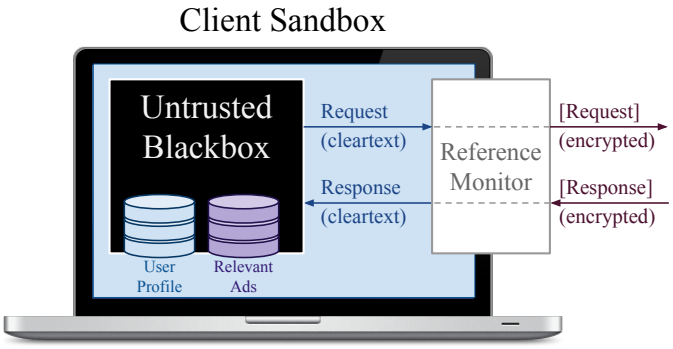


Figure 6.2: The client framework

ports or subscriptions relate to these publishers or advertisers. The dealer then associates these reports with clients. Any clients associated with a threshold number of reports are flagged as fraudulent.

Client Framework

Messages between the client and the broker are encrypted to prevent the dealer from observing their contents. It is critical, however, that users, or privacy advocates operating on behalf of users, are able to verify that the client cannot *undetectably* leak private information in the encrypted messages. Towards this end, the Privad client architecture allows for a thin trusted reference monitor between the client and the network. (Figure 6.2). The reference monitor framework provides users and privacy advocates with a hook to detect privacy violations. The reference monitor validates message contents and performs encryption operations, and ensures that 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 simple, open source reference monitor, which is open to validation so its correctness can be verified, and which, therefore, can be trusted by the user.

Privad architecture allows for multiple competing brokers each with a client on a given user's computer. There could be either independent clients operating in parallel, with each client fully implementing the Privad protocol, building its own user profile, and even communicating with their individual dealers. Alternatively, clients could leverage a number of shared services implementing common Privad functionality and even basic scraping modules. This shared functionality, for instance, could be exposed by browser vendors in order to efficiently support multiple clients. Moreover, dealers could also be shared among multiple brokers.

Ad Dissemination

The fundamental approach to preserving privacy in private-by-design systems is based on prefetching more ads than will be displayed to a user. This must be done in such a way that no entity in the system is able to discover which ads are shipped to which clients. Undoubtedly, the simplest way to guarantee full privacy is to flood all ads to all clients (as this approach prevents the broker from obtaining *any* new information about the clients). However, a measurement study of Google search ads [34] revealed that there are too many ads and too much ad churn for this approach to be practical. This study also found that ad impressions are distributed according to the power law: a small fraction of broadly targeted ads (ca. 10%) receive a large fraction of impressions (ca. 80%). In practical terms, it means that only this small portion of all ads should be delivered to all users (for instance using a P2P mechanism like BitTorrent). Cost effective dissemination of the remaining 90% of ads requires finding a sweet spot between privacy and scalability. Towards this end, Privad employs

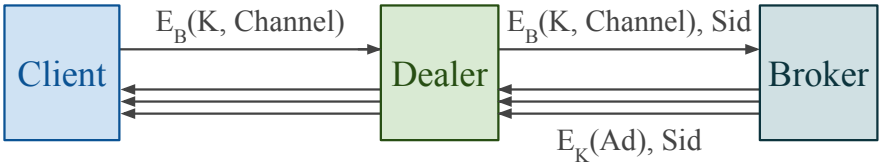


Figure 6.3: Message exchange for Pub-Sub ad dissemination. $E_x(M)$ represents the encryption of message M under key x. B is the broker's public. K is a per-subscription symmetric key generated by the client.

a privacy-preserving publish-subscribe (Pub-Sub) mechanism between the broker and clients to disseminate ads.

The main idea behind Pub-Sub mechanism is to map all ads into generic interest categories and to define subscription channels as an interest category combined with broad demographics (such as geographic region, gender, and language). This must be done in such a way that no sensitive information is leaked in the subscriptions. In other words, channel definitions must be broad enough to accommodate a large number of legitimate subscribers (to guarantee k-anonymity), and yet keep overhead to a minimum to achieve an acceptable scalability. The set of channels is assumed to be pre-defined by the broker and distributed to all clients in advance (i.e., by hosting a signed copy of the complete set of channels at the dealer).

The Pub-Sub message exchange proceeds as shown in Figure 6.3. First, the client generates a request to join a channel. The join request is encrypted with the broker's public key (B) and transmitted to the dealer. The request contains the Pub-Sub channel id (*Channel*), and a per-subscription symmetric key K generated by the client. Key K is later used by the broker to encrypt the stream of ads sent in response to the client. When the dealer receives a join request, it generates a subscription ID (*Sid*). It also stores the mapping between *Sid* and

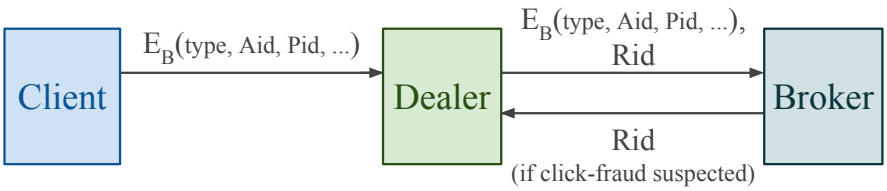


Figure 6.4: Message exchange for view/click reporting and click-fraud defense. *B* is the public key of the broker. *Aid* identifies the ad. *Pid* identifies publisher website or application where the ad was shown. *Rid* uniquely identifies the report at the dealer.

the client, and appends the *Sid* to the message forwarded to the broker. The broker then tags all ads published on this subscription channel with *Sid*, which the dealer uses to look up the intended recipient to forward the ads to.

Since clients generate unique (random) symmetric keys for each subscription, the broker is unable to link multiple subscriptions to the same user and therefore cannot reconstruct interest profile of the user. Additionally, to prevent the broker from correlating subscriptions based on their time of arrival, the system adds some amount of jitter by requiring the clients to arbitrarily delay subscription requests.

View/Click Reporting

In addition to ad delivery, Privad requires a second communication channel between the client and the broker in order to report views and clicks as well as other ad-initiated user actions (i.e., conversions). Reports communicate to the broker the minimum information required for billing and accounting: the type of event (view, click, etc.), the ad id (*Aid*) and publisher id (*Pid*). The *Aid* uniquely identifies the ad, and the *Pid* identifies the advertising inventory (e.g., a webpage, an application) where the ad was displayed. Similarly to the

subscription request, the report is encrypted with the broker's public key and blindly relayed by the dealer (Figure 6.4). As mentioned before, the dealer also can help mitigate click-fraud attacks. Towards this end, for every report the dealer generates a unique id (*Rid*), records the mapping of *Rid* to the client and forwards *Rid* to the broker along with the original message. As a result, fraudulent reports can be traced back to the clients who generated such reports.

In certain cases, the client might have multiple reports to send at once (for example, when multiple ads appear on the same webpage). To prevent the broker from correlating such reports, the client needs to stagger them by adding random delays.

User Profiling

A user's profile consists of a collection of attributes that characterize the user. As mentioned previously, the client software agent is tasked with compiling a profile by monitoring user activity. Privad relies on three basic approaches to accommodate client-side profiling: crawling, scraping, and metadata [33].

Crawling: This approach requires the broker to first crawl the web and preclassify webpages in a way similar to classification performed by existing cloud-based crawlers. The client then can retrieve the attributes associated with a visited webpage from the broker in a private manner. The attribute query protocol is almost identical the pub-sub used for ad dissemination. The protocol follows a simple request-response pattern: the request contains the website URL and the response returns associated profile attributes. The main advantage of this approach is that it allows the broker to employ complex algorithms to assign

attributes to arbitrary content. On the flip side, backend classification clearly will not work for webpages that require the user to log in as well as for offline and desktop applications.

Scraping: There are a number of opportunities for the client software to locally scrape profile information from webpages and desktop applications. These include websites (and applications) that contain structured information, which can be mapped directly to user attributes. For example, user profile can be scraped from online social networks, purchase history from shopping, and travel sites, etc. To facilitate client-side scraping, Privad envisions a modular architecture with website- and application-specific plugins provided and regularly updated by the broker. Since scraping works with websites that require the user to log in, it can be used complementary to crawling. However, mapping unstructured content (e.g. blogs, search terms, text documents) to user attributes on the client requires complex machine learning algorithms. Implementing this functionality in the client alone is not feasible due to the practical limits on the complexity of the client. Instead, a middle ground approach can be taken to address this challenge. In this approach, some pre-processing can be carried out at the client and then the output can be mapped to attributes at the backend using the previously described privacy-preserving querying mechanism.

Metadata: Privad argues that the broker can create additional incentives for website owners to provide the Privad client with profile attributes in the webpage metadata. Similarly, local applications can also communicate profile attributes directly to the client. As an incentive, the broker could offer a portion of the ad revenue to the website or application providing profile information.

This can be done by keeping track of sources of profile attributes leading up to a click and including these sources as a part of the anonymous click report. In effect, this may create a new ecosystem, where websites are rewarded for highly targeted content leading to better profiling, fewer ads and ultimately a better user experience.

Click-Fraud Defense

While neither Privad nor current ad networks have a silver bullet against click-fraud, the private-by-design architecture makes click-fraud defense more challenging. This is the inevitable cost an ad network has to pay when it gives up direct control over the client-side in order to achieve better privacy. Nonetheless, Privad addresses this challenge by proposing a number of techniques to detect a fraudulent client. These range from rate-limiting according to per-client thresholds on the number of subscriptions, and view and clicks reports, keeping historical statistics on per-advertiser and per-publisher performance and looking for anomalies, building honeyfarms that attract and identify click-fraud malware, and dealers blocking compromised hosts that appear in various public blacklists.

One novel technique first proposed by Privad and later extended in [35] is termed "bait ads". Bait ads can be intuitively described as CAPTCHAs for ads – they contain targeting parameters that are completely unrelated to the actual content (ad body, graphics, flash animation) displayed to the user. For example, a bait ad may advertise a "dog collar" to "cat lovers". Normally, such mistargeted ads should produce extremely low click-through rates. However, a bot would trigger multiple baits, as it would be unable to distinguish them from

normal ads. As a result, an unusually high rate of clicks on bait ads would help verify that an attack is underway and also help identify the fraudsters. Bait ads and other click-fraud detection mechanisms complement each other: deployed in parallel they will significantly raise the bar for an attacker. A more detailed discussion of click-fraud defense in Privad can be found in [32].

Once an ongoing click-fraud attack is detected, identifying and blocking fraudulent clients is straightforward. To do so, the broker notifies the dealer of the *Rid*'s of reports suspected of being involved in click-fraud. The dealer traces the *Rid* back to the client. If the dealer receives a threshold number of notifications for a given client, it blocks subsequent reports from that client. Moreover, the dealer itself can monitor clients to see if they have an unusually high volume of views or clicks, and flag them accordingly.

Reference Monitor

In order to allow for proprietary closed-source client agents, Privad provides a sandboxed environment with a trusted reference monitor. The reference monitor is the only communication gateway between the client and the server-side components in the Privad architecture. In other words, only the reference monitor is allowed to perform network I/O. It exposes a thin set of APIs to the sandboxed client, allowing it to generate symmetric subscription keys, encrypt and decrypt messages, and communicate with an authorized dealer. This API set is designed to be extremely small so that the correctness of the reference monitor can be easily verified.

The reference monitor validates message contents to ensure that the client adheres to the protocol specifications. It also ensures that no sensitive information is leaked in the message content or through covert channels. For example, by having the monitor generate keys and perform the encryption Privad reduces the possibility of the client passing information through random bits in generated keys, or through the randomized padding in the encrypted message. Additionally, the monitor is allowed to add random delays or jitter to further reduce the possibility of exploiting timing as a covert channel. For this reason, Privad is designed to be delay tolerant – all operations are asynchronous, and no message requires an immediate response. Finally, the reference monitor provides a hook for auditing the client software.

6.2 PDDP Overview

Privad is carefully designed to provide only the minimum information needed by the broker, advertisers, and publishers to run the ad business: requests for ads and anonymous reports of clicks and views (i.e., which clicks and views occurred on which ads at which publishers). This minimum information, however, is not sufficient if the goal is to get deeper insight into user behavior and system performance. For instance, key players may want to know more, in the aggregate, about what activity leads the profiler to detect an interest in the first place. They may wish to know the level of interest, or correlations between interests. They may wish to know whether certain publisher websites led to better conversions, and so on. With centralized (non-private) tracking, this information is all available locally, and can simply be mined. With Privad, this information is all tucked away on user computers, which precludes broad statistical

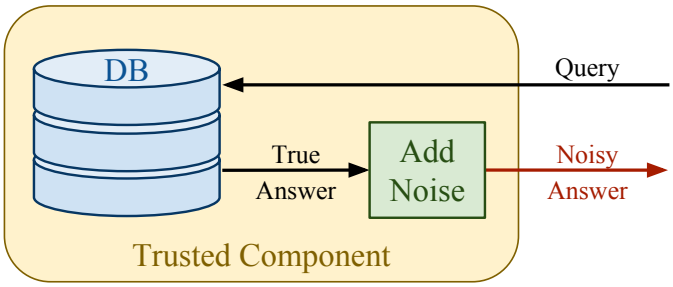


Figure 6.5: Traditional deployment model for Differentially Private systems

analysis of user data. Moreover, once Privad is deployed, it would be virtually impossible for the system designers themselves to debug the system without infringing on user privacy. To address this issue, Privad needs to provide support for privacy-preserving statistical queries over distributed user data.

One approach to supporting privacy-preserving statistical queries is to add noise to the answers of queries, in such a way that the privacy of individual users is protected. An instance of this approach popular in the research community is *differential privacy (DP)* [23, 24]. Specifically, DP adds noise to the answers for queries to statistical databases so that the querying system cannot infer the presence or absence of a single user or a set of users. DP provides a provable foundation for measuring privacy loss regardless of what information an adversary may possess.

The traditional deployment model for DP assumes a centralized database (see Figure 6.5). The system operating the database is trusted with its content, and is also trusted to add noise to the information released from the database. The private-by-design advertising scenario is different in several respects. First, there is no trusted centralized database; individual clients maintain their own data. Second, the information is distributed among potentially millions of

clients. Therefore, the private-by-design advertising settings call for a practical mechanism that provides some form of *distributed differential privacy*.

As it turns out, the existence of the dealer in Privad, trusted not to collude with other components of the system, can be leveraged to accommodate the Practical Distributed Differential Privacy (PDDP) system [19]. Figure 6.6 shows how PDDP can be deployed on top of the Privad dealer. Now, an analyst (e.g., a broker, an advertiser), who wishes to make statistical queries over some number of Privad clients, can formulate a query and transmit it to the dealer, which in turn forwards it to the required number of clients.

Each query comes with a number of buckets that specify possible answer ranges. A client locally executes the query, and for each bucket it produces a binary value indicating whether the query result fell within the range of that bucket. Then, the resulting bit vector is encrypted with the analyst's public key (using Goldwasser-Micali bit-cryptosystem [30]) and uploaded to the dealer. Meanwhile, the dealer and clients, using the XOR homomorphic property of the GM cryptosystem, *collaboratively and blindly* generate noisy answers that mimic a number of additional client responses to produce the required amount of differentially private noise. Finally, the dealer mixes received client answers with noisy answers, and forwards everything together to the analyst, who then decrypts the received answers and computes the statistical result under the differentially private guarantee.

A major issue with DP in practice is that systematically repeated queries can be used to eliminate the noise and reveal the true answer. Traditional DP systems deal with this through the notion of a *budget*. Each query deducts from the budget, and when the budget is spent, the additional queries are simply

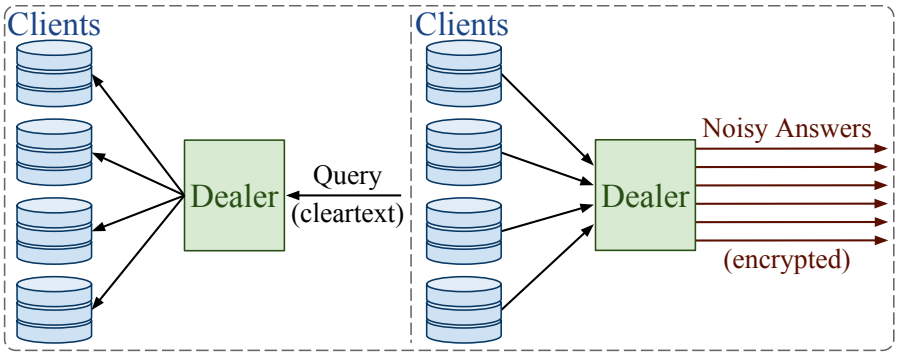


Figure 6.6: Differential privacy in the context of private-by-design advertising

not allowed. However, this approach is not practical in the advertising context, which requires longitudinal analytics. Rather than setting a hard limit on the cumulative privacy loss, PDDP treats it as an ongoing measure referred to as the *privacy deficit*. The notion of the *privacy deficit* makes it possible to address a number of open issues. First, it provides a quantitative measure of the privacy cost incurred as a result of a meaningful statistical analysis of the user population. Second, as described in Section 9.5, the accumulated deficit can be analyzed under a worst-case scenario to determine whether it theoretically allows a malicious analyst to discover a number of sensitive user attributes. Indeed one question we address in this study is "How far from actual reality is differential privacy's worst-case model?"

6.3 Related Work

In this section, we describe previous attempts at building and deploying a private-by-design system.

Among the systems cited in Section 1.2, two were never implemented: PiCoDa [49] used simulations to evaluate timing performance for the protocol data, MobiAd [36] deferred the effort required to build and deploy a prototype to future work. Only three systmes (Adnostic [58], RePriv [29] and Privad [32]) have been built as a functional research prototype.

The core targeting system of Adnostic [58] is available as a Firefox addon. The user profile in this implementation is a weighted list of categories derived from *Google Ads Preferences*. In order to build a profile, Adnostic monitors user browsing activity and for each visited page assigns a number of most relevant categories. These categories are then aggregated in the user interest profile with weights reflecting number of page visits, number of clicks and the page viewing duration. The categorisation is based on computing similarity scores between the webpage metadata and interest categories, which uses a pre-computed matrix of cosine similarity between category words and most common bookmark tags from delicious.com. While the addon also contains ad rendering and even ad scraping functionality, these modules were only used to compute benchmarks reported in the paper. Beyond that, unfortunately, the system was never deployed and evaluated with actual users.

RePriv [29] describes a research prototype, built on top of C3, an experimental browser developed in .NET [17]. In RePriv, the user profile is also constructed locally in the browser by mining user browsing data to infer personal interests. Additionally, RePriv allows service providers to register verified site-specific miners, thereby improving the quality of inferred information. The release of sensitive information is controlled by the user, who determines how much private information can leave the browser, and what exactly is shared

with each party. Once explicit user consent has been granted, RePriv sends relevant portions of the user profiling information directly to the ad network. Functionality implemented in the prototype includes a behavior mining algorithm, a communication protocol for secure dissemination of profile information, and an extension framework for loading third-party software that utilizes user profile. Behavior mining is based on classification of visited webpages using a hierarchical taxonomy of document topics derived from the Open Directory Project (ODP).[1] For each visited page RePriv assigns topic probabilities using a Naïve Bayes classifier trained over a set of documents from each category of the first two levels of the ODP taxonomy. Classification information for each page is stored locally alongside with the browsing history. Profile information exposed by RePriv consists of a list of taxonomy categories together with an indication of the interest level for each category (computed as a fraction of browsing history classified with that category). However, the authors do not elaborate on the extent this information can be leveraged for ad targeting, neither do they report any results regarding deployment or evaluation in the advertising context.

A proof-of-concept implementation and a pilot deployment of Privad was described in [32]. In this experiment, the client component was distributed as a stand-alone Firefox addon to 2083 Mechanical Turk[2] workers, who were remunerated for having the addon installed for at least one week. User profiling in the prototype implementation was based on scraping users' Facebook profile and Google Ads Preferences. However, the extracted profile information was never used, the system simply scraped and republished Google ads without any targeting. Overall, the system was in continuous operation for a year, retaining a fifth of its original user base. During this time there were 217K ad views, 238

[1] http://dmoz.org
[2] https://www.mturk.com

ad clicks. Admittedly, the main goal of that deployment was to evaluate technical aspects of the architecture, and as such it provides little insight into the advertising utility of the system. In contrast, the purpose of this work is to exercise the private-by-design advertising in realistic settings. Towards this end, we bring profiling and targeting several steps closer to that of a commercially deployable system. Our prototype does real targeting based on user searches and products browsed on shopping sites, and delivers real ads by pulling them from online shopping APIs. Moreover, alongside with the advertising system, we deployed a private analytics component providing a deeper insight into user behavior and system performance.

CHAPTER 7

PROTOTYPE DETAILS

While Privad establishes a basic private-by-design architecture described in the previous section, to put together a fully functional ad system prototype we had to fill in a number of gaps. Due to the experimental nature and small scale of our system, we cannot work directly with advertisers or ad networks. Instead, we use product information from major shopping engines to as a proxy for creating Privad ads. Given such product-oriented ads, for profiling and targeting we focus exclusively on the user purchasing intent. We rewrite Google ad iframe requests and repurpose resulting adboxes to render Privad ads, which prevents exposing users to more ads than they would normally see.

In the rest of this chapter, we describe in detail the challenges we tackled while building a private-by-design ad system prototype without support from the ad industry. We also discuss ways in which the private system design evolved to meet practical concerns. Then, we report privacy issues identified and addressed along the way. Finally, we describe implementation details specific to our prototype.

7.1 User Profiling

Our approach to profiling combines aspects of both *crawling* and *scraping* described in Section 6.1. Since we can only generate product-related ads, our main goal with respect to user profiling is to identify and capitalize on the user transactional (purchasing) intent [38]. Therefore, we focus on two main signals: user browsing activity on shopping websites (product-based targeting) and product-

related searches on major search and shopping engines (search-based targeting). Towards this end, we compiled a whitelist of shopping websites containing almost 14K entries by crawling retailers on Shopping.com that also appear in Alexa's top 1M [1]. The whitelist also includes Alexa's top 500 websites for a number of top-level shopping categories (such as "Clothing", "General Merchandise", "Gifts", etc.), as well as the 5 largest search engines. Additionally, we built a dictionary of product-related terms by crawling a random set of ca. 80 million products offered on Google Products, Amazon and Shopping.com. From the collected set of 10M terms appearing in product titles, we removed terms with fewer than 100 occurrences, resulting in 180K whitelisted keywords.

Each time a user issues a search query on one of the whitelisted websites, Privad client captures the query, extracts search terms and filters stemmed keywords through the product-related dictionary. The resulting list of keywords is then used to establish an ad channel. Additionally, the identified keywords are cached and used to deduplicate future product searches.

We profile user shopping activity by monitoring browsing behavior on whitelisted websites and applying customized scrapers to identify specific products the user is interested in. The fundamental challenge of this approach is the effort required to build specialized product scrapers for the majority of popular online retailers. In our experimental prototype, we sidestep this challenge by leveraging scraping functionality developed by InvisibleHand [7]. InvisibleHand tries to identify a product the user is browsing for, and then displays a notification if there are better deals available for the product. At the moment, InvisibleHand scrapers identify products on 670 shopping websites. As described in Section 7.6, neither the whitelists nor the scraping functionality is hard-coded

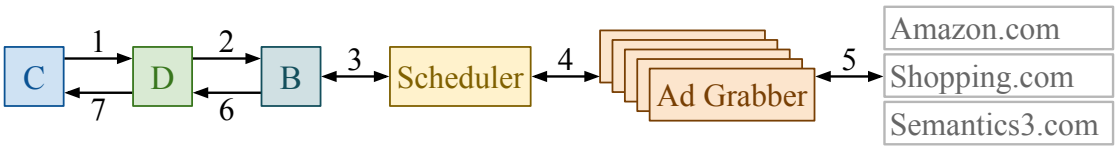


Figure 7.1: Ad generation pipeline. C is client, D is dealer, B is broker

in the client. Instead, clients periodically check with the broker and download updated lists of scrapers and whitelisted domain names as soon as they become available.

7.2 Ad Generation

The experimental nature of our system dictates that we neither work with any commercial advertising companies nor generate any revenue by displaying ads. Instead, we create mock-up[1] ads using product listings from three major shopping engines: Amazon.com, Shopping.com and Semantics3.com.[2]

The series of steps performed to generate Privad ads is shown in Figure 7.1. Once the Privad client detects a new product or product-related search, it uses product title or whitelisted search keywords to establish a new interest channel and request ads from this channel. We do not rely on any predefined interest hierarchy to map profiling information to channels. On the contrary, the interest channels are generated at runtime and are fully defined by the associated product information. As such, we distinguish between two channel types according

[1]The generated ads look like legitimate Google ads and link to real products. We call them 'mock-up' only because they are not handcrafted.

[2]While we initially started out with Google.com/shopping as our third product provider, over the course of the prototype development Google's Search API for Shopping was deprecated and then eventually sunset.

to targeting parameters (product and search targeted channels). Targeting information associated with each channel is used to request a number of ads from the broker in an anonymous, privacy-preserving manner. The ad request (step 1 in Figure 7.1) is relayed by the dealer to hide the client's identity. The encryption mechanism described in Section 7.4 is used to prevent the dealer from eavesdropping.

The dealer batches requests from multiple clients into a single RPC request, which is uploaded to the broker once every 30 seconds (step 2). Upon receiving an ad request, the broker forwards targeting information to the *ad grabbing* service (in step 3), which then uses it to make a product search on one of the three shopping engines (steps 4 and 5). Up to 20 most relevant product offers from the result set are converted into textual ads and returned to the broker. The conversion is straightforward and consists of removing stop words and excessive punctuations from the product title and description, distributing the remaining terms together with a price tag over ad head and body (25 and 70 characters long), and adding a short display URL. Most of the auto-generated ads produced this way are intelligible and look almost indistinguishable from AdSense ads. However, in some cases products-to-ads conversion fails to produce meaningful output (see examples in Figure 7.2). Finally, the broker bundles the resulting set of text ads into a single ad channel and ships it back to the dealer (step 6). The ad channel is stored at the dealer until it is eventually retrieved by the client (step 7).

Shopping engines proved to be the major bottleneck in the ad generation pipeline (with requests to Shopping.com taking on the order of several seconds). Additionally, they tend to impose a limit on the number of allowed search re-

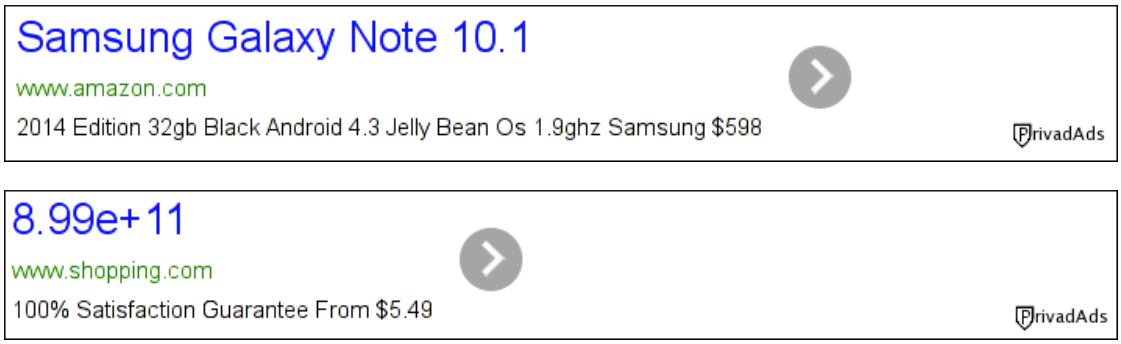


Figure 7.2: Examples of auto-generated Privad ads

quests per IP. To address this scalability challenge we replicated the ad grabbing service on a number of machines and placed a simple round-robin load-balancer (Scheduler in Figure 7.1) in between the Ad Grabbers and the broker.

7.3 Ad Selection and Placement

Since we lack real publishers, we render Privad ads in the existing Google AdSense adboxes (i.e., adboxes that contain contextual ads and appear on publisher websites). While we modify Google ad frame requests to display more textual advertising (as opposed to flash and image ads), we avoid exposing users to more ads than they would normally see. Google allows publishers to specify style parameters for textual ads so that ads have the same look and feel as the publisher website. Instead of fully re-writing adbox html code, we leverage this functionality by identifying and surgically replacing only relevant ad content (head, body, click URL, etc.). As a result, apart from a small logo indicating a Privad adbox, Privad ads look almost indistinguishable from AdSense text ads. However, while preserving ad style preferences, this approach is ad hoc in nature and depends on cues in the html code characteristic to various

ad elements; once html code is modified our ad serving modules might fail to render Privad ads.[3]

Our client implementation allows us to experiment with different placement strategies: control mode (only Google ads), full-on Privad mode (only Privad ads), mixed mode (a mixture of Privad and Google ads in the same multi-slot adbox) and random mode (uniform distribution of adboxes filled with either Privad or Google ads). Additionally, our client respects user preferences and does not request or display any ads in Private Browsing Mode, since uploading ad requests or view/click reports in PBM would clearly violate user expectations. Also, it does not display Privad ads if it detects any adblocking mechanisms (browser addons or DNS-based blocking).

Unfortunately, we lack bid information to run a full-fledged second price auction. Instead, we conjecture that the click probability, and hence the user score [54], is inversely related to the amount of time passed since an interest was detected. In other words, the *click probability* on an ad from a particular channel *decreases over time*. To verify this observation, the Privad client allows experimentation with three ranking mechanisms according to the age of an ad channel: most recent first, uniform random and binomial (pick the most recent with probability 1/2, second most recent with 1/4 and so on).

7.4 Message Exchange

Following the original Pub-Sub model [32], the dealer uses an asynchronous relay protocol to forward messages between the clients and the broker. Each new

[3]Unluckily, as described in Section 8.1, this is exactly what happened during our deployment.

Privad client is bootstrapped by requesting a unique client id from the dealer. This Uid then is sent to the dealer alongside the encrypted message payload. For every incoming request that requires an explicit reply from the broker, the dealer generates a unique request ID (*Rid*). It replaces Uid with *Rid* in the client's request and also stores the mapping between them. Finally, multiple client requests are batched together and uploaded to the broker at regular intervals. The broker then attaches the *Rid* to every response it sends back, which the dealer uses to look up the intended client and save the broker's response tagged with the client's Uid. A client uses its Uid to periodically poll the dealer for any new messages from the broker.

As opposed to the original Privad design, which relies on a predefined set of channels to map user attributes to ads, in our prototype channels are established on the fly in response to an ad request (as long as the result set of the corresponding product search is non-empty). This allowed us to reduce the original Pub-Sub ad dissemination mechanism to an asynchronous request-response, which operates as follows. Client requests take the form $(E_{PK_{broker}}(K_{shared}), E_{K_{shared}}(request))$ where the actual message payload is encrypted with a randomly generated 128-bit AES key K_{shared}, and the symmetric key itself is encrypted with the 1024-bit public RSA key of the broker PK_{broker}. Randomized padding is added to defend against dictionary attacks. The response from the broker is then $E_{K_{shared}}(reply)$ encrypted with the symmetric key from the corresponding request. The request and response messages of this form serve as building blocks for all client-broker communications, which include ads delivery, click and view reporting, as well as the new communication channels required for distributing scrapers, website and keywords dictionaries and experimental configurations (Section 7.6).

7.5 Privad Implementation

We built a fully functional Privad system. Following the architecture described in Section 6.1, our prototype comprises three principal components: the client, the dealer and the broker. The client is implemented as a 154KB Firefox addon written entirely in JavaScript (8.5K lines of code not counting the pidCrypt library and autogenerated RPC client code). All backend components are implemented in Java, totaling 14K lines of code with the dealer, broker, and ad generation infrastructure taking roughly equal parts. All Privad datatypes and interfaces between system components are defined in 600 lines of Apache Thrift IDL [3], producing 48K and 6K lines of Java and JavaScript code respectively.

We chose to implement the client as a browser addon to enable us to scrape highly-dynamic AJAX web applications, which would have been impossible with a standalone daemon or local browser proxy. Concerned with Javascript performance for cryptographic operations, we delegated all CPU-intensive processing to two independent web workers (one responsible for Privad-related functionality, the other for PDDP). These background Javascript threads have no access to the DOM and communicate with the main browser thread via asynchronous message passing. Also, they serve as the single gateway between the client and the dealer. By outsourcing cryptographic operations, data serialization and network communication to background workers, we ensured that there was no negative impact on the user's browsing experience.

These web workers expose a thin set of predefined API and thus serve as a proof-of-concept reference monitor that performs the encryption and network I/O, but does not validate message contents. Potentially our client could get

around this reference monitor and establish additional communication channels to the broker (in that sense the web workers are not the only point of communication between the client and the backend components). But in contrast to the proprietary closed source clients anticipated by Privad, our implementation is *open source*. It can be easily validated through manual inspection[4] and, therefore, does not have to be sandboxed.

All client-dealer communication is performed over HTTP to accommodate clients behind firewalls and proxies. We use JSON since it is the only format currently supported by the JavaScript Thrift library. While the dealer is written as a Jetty [57] server handler, other backend components are built on top of Thrift servers and communicate using binary Thrift format.

7.6 Client Details

In addition to the two communication channels between the client and the broker (ad delivery and view/click reporting) required by vanilla Privad, we also introduced a distribution and update mechanism for product scrapers, shopping websites and product term whitelists. To keep their whitelists and scrapers up-to-date, clients periodically issue and upload a request containing the hash of the currently active whitelist. As long as the hashes of the client's and broker's lists match, the request is ignored by the broker. If hashes do not match (e.g., entries were added or removed from the whitelist), the new whitelist is sent in response.

[4]Which is the case – our client code has been vetted by Mozilla reviewers

A similar mechanism is used to disseminate experiment configurations. In addition to the hash of the configuration in place, clients also include their configuration class in the request. This class is randomly selected from 16 available values during the client's first launch. By dividing client's population into 16 groups, we are able to run multiple experiments in parallel. An experiment configuration contains a number of parameters that specify start and end timestamps and regulate ad placement strategy (none, everywhere, mixed, random), channel ranking mode (random, most recent, binomial), targeting mode (product-based, search-based, random) and various channel-related attributes (max channels in place, max channel lifetime, max view opportunities, etc.).

7.7 Practical Privacy Issues

Running a 'real' ad system requires a number of functions not anticipated in Privad's design. Consequently, to build an operational private-by-design system, we had to address several practical privacy-related concerns that arose as a result of the added functionality.

Search terms in ad requests. The original Privad design envisioned that relatively broad product or interest categories would be conveyed in ad requests. In practice, however, we had to use search terms and product names derived from potentially error-prone web page scraping. To mitigate possible privacy loss through these search terms and product names, we implemented the whitelist described in Section 7.1. In spite of this, in rare cases, it may be possible to identify users through the ad request. This is an unanticipated problem that needs further consideration.

Timestamps in ad requests and reports. In order to select relevant configuration parameters when serving a client's ad request, the broker needs to know both the client's configuration class and the timestamp at which the request was made. Moreover, view/click/conversion reports are also timestamped, which allows us to find the delay between the interest detection and the subsequent view/click/conversion events as well as the temporal distribution of these events. However, revealing unobscured client timestamps constitutes a major privacy leak. First, the broker may exploit these timestamps and try to fingerprint clients based on their clock skew. Second, sending timestamps in the client's local timezone breaks channel unlinkability (e.g., when there are very few online users in a particular timezone sending view/click reports for ads from different channels).

We prevent this privacy leakage by uploading client timestamps converted to the same timezone (UTC) across all clients. To be able to compute event distribution over time, we add a timestamp in the client's timezone and an event subtype (ad request, view, click, conversion) as the meta-info of the encrypted message uploaded to the dealer, and store it there without forwarding it to the broker. To hide potential clock skew, we currently use timestamp granularity of 5 minutes. Additionally, it is possible to add some amount of noise to the timestamps, introduce longer (currently only 30 seconds) upload cycles and jitter (i.e., delay random messages for several upload cycles) at the dealer.

Publisher info in view reports. Using the publisher domain from the view reports, the broker can track all websites where an ad (or ads from the same channel) were displayed, as long as ad and channel ids are unique across all clients. However, in a commercial Privad deployment the broker must not gen-

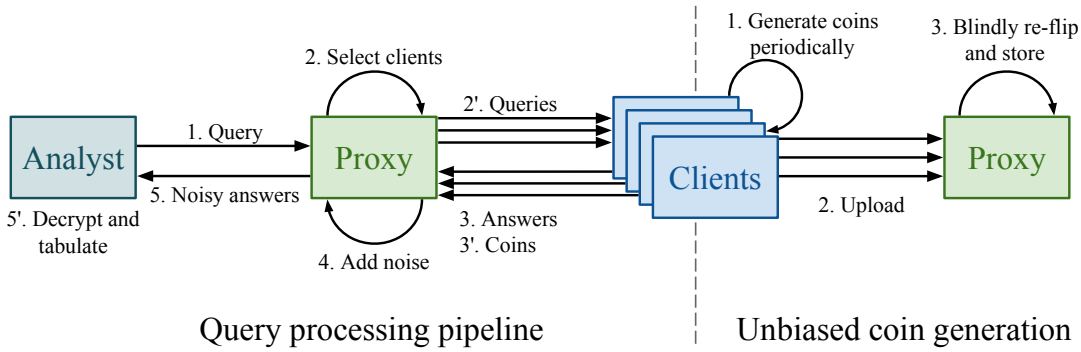


Figure 7.3: PDDP - the private analytics system

erate unique ids for subscriptions to the same channel, and it should be easy to detect when this assumption is violated. The problem will come up only when there are very few users subscribed to a channel. Reporting publisher domain together with a view timestamp also breaks channel unlinkability (e.g., when there are multiple adboxes on a page filled with ads from different channels). In our prototype, only click and conversion reports contain publisher info. We break the view report into two parts, one containing ad specific information and the other publisher data, and upload them to the broker independently with a random delay.

Adbox id in view reports. To discover which ads are displayed together, a randomly generated adbox id is included in view reports. Unlinkability then can only be ensured, if we populate each adbox exclusively with ads from a single channel.

7.8 PDDP Implementation

Following the design described in [19], we built the PDDP private analytics subsystem by retrofitting Privad components with additional PDDP functionality. In our implementation, the broker takes the role of an analyst, the dealer acts as the PDDP proxy, and the Privad addon as a client. The query processing funnel contains the following steps (as illustrated in Figure 7.3). First, the broker submits a PDDP query to the dealer (step 1). A query includes a number of SQL statements, buckets definitions (ids, and lower and upper bounds), privacy parameter ϵ, and start and end timestamps. Additionally, it can specify the required number of answers (no less than 10) and the target configuration class. The dealer verifies that the query does not modify the client database (queries containing keywords like 'create', 'pragma', 'delete', etc. are rejected),[5] and adds it to the list of pending queries.

For every pending query the dealer maintains a set of clients who already uploaded an answer to the query. Clients periodically poll the dealer and retrieve a new (random) PDDP query that they have not yet answered (step 2). We allow clients to process only one query at a time to avoid overloading the user machine and adversely impacting browsing experience. Thus, a client can request the next query only after it has finished executing the current query and uploaded the answer.

Upon receiving a query, the client executes it over its local database and produces a list of numerical answers, which it then maps to buckets by assigning a '1' or a '0' to each bucket, depending on whether or not one of the answers fell

[5]However, we allow PDDP queries to store intermediate results as key-value pairs in a dedicated table, which is wiped clean after every query execution.

within the range of the bucket. Then the client encrypts each per-bucket binary value using the broker's Goldwasser-Micali (GM) [30] public key. The resulting set of bucket ids together with encrypted bits make up a PDDP answer that is submitted to the dealer (in step 3).

After receiving a client's answer, the dealer validates the answer (Jacobi symbol of a valid GM-encrypted value equals to '+1') and stores it locally. Once the dealer collects the required number of answers or the query expires, it adds a number of randomly flipped bits or *coins* to each bucket to ensure differential privacy (step 4). Given privacy parameter ϵ and the number of answers c, the minimum number of per-bucket coins required to achieve (ϵ, δ)-differential privacy is [19]:

$$n = \lfloor 64 \ln(2c)/\epsilon^2 \rfloor + 1$$

Finally, the dealer shuffles clients' answers and random coins together and uploads the resulting set of (bucket id, encrypted bit)-pairs together with the value of n to the broker. Upon receiving this message, the broker decrypts and sums up all binary values for each bucket id (step 5). It then subtracts $n/2$ from each per-bucket sum to compute a (noisy) per-bucket count (i.e., the number of clients that fall within this bucket, under the guarantees of differential privacy):

$$count = \sum_{i=1}^{c+n} bit_i - n/2$$

By answering a PDDP query a client ultimately reveals a bit of private information, which over a large number of trials can potentially allow an attacker to average out noise and discover private user attributes. In other words, each PDDP query has an implicit privacy cost associated with it, which clients pay when they answer the query. Over time, this leads to accumulation of a *privacy deficit* (i.e., privacy loss across all queries). We keep a record of the per-client

privacy deficit at the dealer. To err on the conservative side, we make no assumptions about possible correlations between buckets and effectively treat individual buckets as separate queries bundled together. Thus, for every client that contributed an answer the dealer adds

$$(\epsilon, 1/c) \times \textit{number of buckets}$$

to the client's privacy deficit. While this is an overly pessimistic approach to tracking deficit, it allows us to have both overlapping bucket ranges and queries producing multiple results that are mapped to multiple buckets. To further reduce the privacy cost, we implemented PDDP queries keyed on experimental configuration classes to target only relevant groups of users.

Dogfooding PDDP enables us to collect various advertising related metrics that cannot be conveyed through Privad without breaking its privacy guarantees. For example, using PDDP we can find per-user click-through performance of Google AdSense ads and compare it with Privad's. Moreover, using various client-side stats, PDDP allows us to peek beyond simple views and clicks and analyze user engagement with the advertising content.

In addition to storing information related to core Privad functionality (experimental configuration, captured searches and products, ad requests, active ad channels, view and click stats, etc.), we also collect a number of additional metrics. These include performance stats for several types of Google ads (text, banner, flash), user engagement (time spent actively browsing a landing page), browsing session and click-chains (series of visited URLs after an ad click). Our client also captures user's shopping activity (products placed in the shopping cart, purchases made), browsing, and bookmarking behavior. Additionally, we store general user information including geographical location and timezone,

OS, language, adblocking addons, as well as overall browser usage. All captured information is stored in a local SQLite database using Firefox's Storage API, thus allowing the PDDP analytics system to query for that information in a differentially private manner.

CHAPTER 8

LARGE SCALE DEPLOYMENT

One major challenge in deploying the Privad prototype is incentivizing users to install it. Since Privad does not provide immediate tangible benefits for the end users, the most viable deployment model is bundling with existing freeware applications with a well-established user-base. In this chapter, we describe our experience in deploying Privad by bundling it with a popular Firefox addon and present various deployment statistics collected by the backend servers.

8.1 Deploying Privad at Scale

The backend component of our Privad deployment contained 9 servers: one dedicated to each of the dealer, the broker, and the scheduler and 6 ad grabbing replicas in order to avoid hitting the per-IP limit on the shopping API request rates.

Deploying the client component of our Privad prototype proved to be a major logistical challenge. First, it required finding addon developers with an established user base (of at least 10K daily active users), who were actively supporting and maintaining their addons and who were also willing to bundle Privad client as a part of their addon. Second, to make the bundle automatically available to users as a part of the addons update mechanism we had to pass the Mozilla review process. Finally, we had to craft an appealing request to prompt the users to opt into the study. Overall, we successfully deployed the Privad client by bundling with two Firefox addons.

In a pilot deployment, we bundled Privad client with iFamebook [6], an addon with 10K daily active users that shows visitors to a user's Facebook profile. During an 8 week deployment, 40K users installed the bundle, 2800 opted into the experiment, and at peak we registered 500 online users. Overall, our pilot deployment generated 315K views, 162 clicks and 2 conversions. We believe that the low opt-in rate was caused by an inconspicuous participation request, which was presented to the users as a drop-down notification bar at the top of the browser window. This notification bar did not retain focus and could be easily ignored by the users.

Second time, we deployed Privad client by bundling it with Google Docs Viewer[1] [5] – a Firefox addon that uses Google Docs to render online documents (pdf, doc, ppt, etc.) in the browser without downloading them. When a user right-clicks on a document URL, this addon appends a context menu entry to open selected document using Google Docs Viewer [4]. We decided to bundle with Google Docs Viewer mainly because the addon is actively supported and extended, and therefore maintains a sizable population of almost 80K daily active users. In an attempt to improve the opt-in rate, we modified the participation request (shown in Figure 8.1) to be displayed in a modal dialog, which does not allow switching browser windows until it is closed by clicking one of the buttons.

Experimental ethics. Participation in the Privad experiment follows an opt-in model.[2] When users update their addon to the version containing the Privad bundle, they are presented with a participation request dialog and are free to choose to join the experiment or not. The participation request contains a

[1] No affiliation with Google Inc.

[2] Mozilla's No Surprises policy requires a opt-in with non-default user action to activate an *"unexpected feature"*, such as Privad.

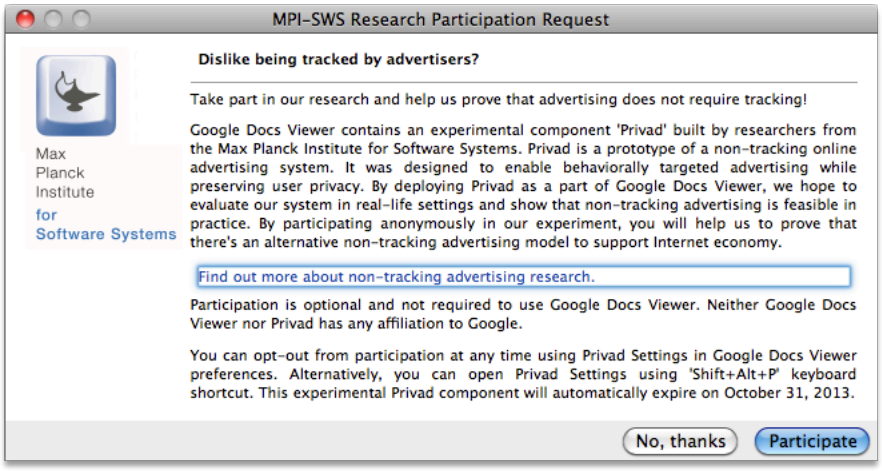


Figure 8.1: Privad's participation request

link to a webpage, which provides a comprehensive description of the experiment and informs the users that a fraction of the Google AdSense ads will be replaced with Privad ads during the experiment. Each adbox containing Privad ads is clearly labeled with a distinct PrivadAds icon, which when clicked leads to the experiment homepage. In case an opted-in user is no longer willing to participate in the study, the Privad client provides an easy way to opt-out. Additionally, Privad honours adblocks and PBM, everything is wiped clean if the plugin is removed.

The system was in continuous operation for more than two months in September and October 2013 with a two-week gap during which it did not serve any Privad ads. This hiccup was caused by a major Google AdSense redesign [8], which changed the html code that ad servers produce to display AdSense ads. As a result, Privad's ad rendering modules were no longer functioning properly and we had to push an update to address issues triggered by the new AdSense design.

Overall, 13K users opted into the study[3] and after an initial bootstrapping period the system was used daily by over 4800 users on average, with more than 2000 users online at peak. In October alone, the Privad backend received 1.1M ad requests, generating 960K channels with 9.5M ads in total. We registered 790K ads views, 417 ad clicks and 4 Amazon purchases (including a "Flower Power Hippie" Halloween costume, among others). During that time the dealer served on average 7.9M daily RPC requests, and forwarded 950K messages from clients to the broker and 60K messages in the reverse direction on a daily basis. In terms of the network utilization, this corresponds to 1.2 GB and 115 GB of daily traffic received and sent to clients. On average, the broker processed 32K search-based and 5.5K product-based daily ad requests, generating 280K and 39K ads respectively. At peak, the load on broker reached 286 requests per second.

To measure the communications overhead at the clients we parsed server-logs generated by the dealer in October and selected 6.5K Uids that appear in logs on at least 7 different days. Figure 8.2 plots the distributions of per-client daily volume of messages exchanged with broker, including ad requests and reports, as well as the daily bandwidth consumption. While detailed information logged by the dealer must not be directly available to the broker, the aggregate stats presented here can safely be made publicly available as part of a monthly operational summary. Surprisingly, we found that the median value for the number of daily ad requests is around 3.2, 11% of the users in our sample never requested any ads, and almost 60% did not generate any ad views. To uncover

[3]No pings are sent to the backend before a user has opted in, as a result we do not know the exact opt-in rate. But based on the number of daily Google Docs Viewer users, we estimate it at around 1 in 15.

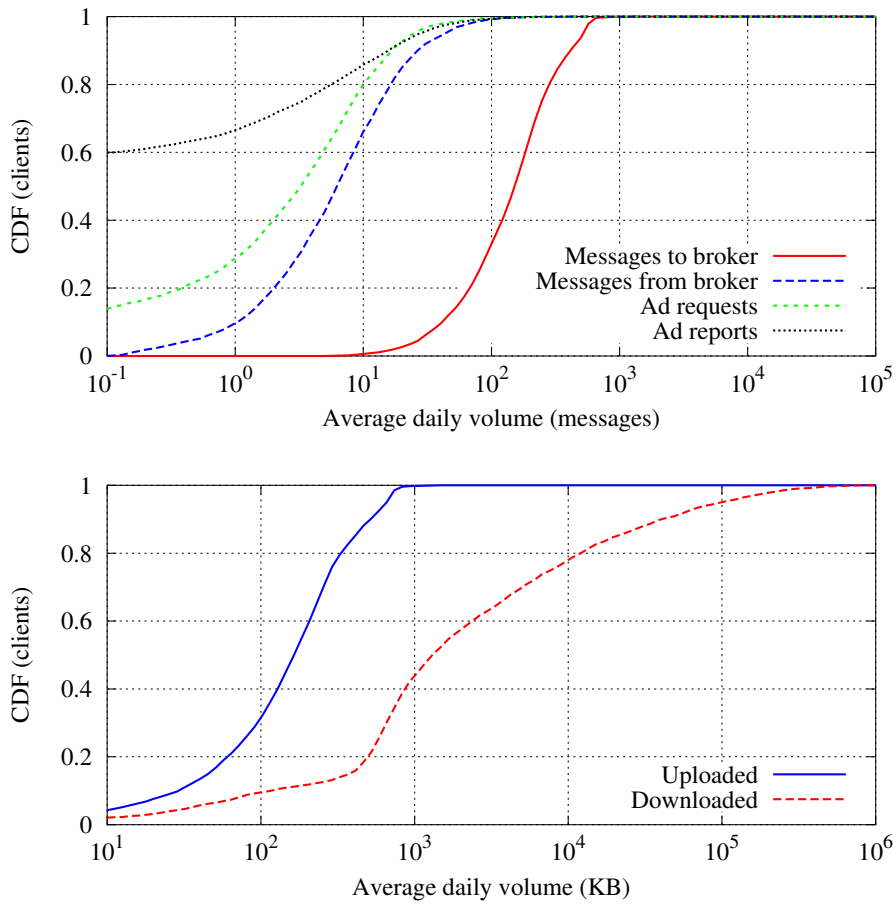


Figure 8.2: CDFs of per-client daily communications overhead

the reasons for the observed behavior, we turned to PDDP analysis, as described later in Section 9.1.

8.2 Privad Advertising

In this section, we report various advertising related stats computed using reports collected by the Privad broker (i.e., not with PDDP). In total, 87% of all requests generated a channel (with 89% and 79% for search- and product-based requests respectively), producing on average 9.8 ads per channel. Overall, we

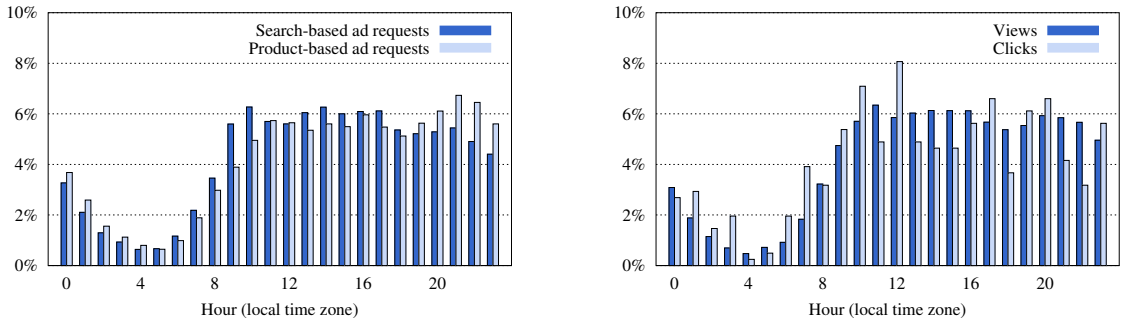


Figure 8.3: Hourly distributions of ad requests, views and clicks

calculated Privad CTR at 0.05%. While this value may seem discouragingly low, as we discovered using PDDP analysis (see Section 9.2), in terms of advertising performance Privad ads are comparable with text ads on Google Display Network.

With slightly more than 400 clicks, we did not observe stark variations in CTRs corresponding to different configuration parameters (e.g., placement mode, channel lifetime, channel selection mode, etc.; all produced roughly equal CTRs). However, we found that the CTR for product-targeted ads is 2.6 times higher than that for search-targeted ads (0.12% versus 0.046%). We also found that more than 70% of the Privad clicks were registered in single-slot adboxes (with corresponding CTR of 1.1%), with another 14% in the first position in multi-slot adboxes (where we observed an exponential decrease in CTR in lower positions).

Figure 8.3 shows hourly distributions of ad requests (both search and product based), views and clicks. To plot this distribution we used event timestamps recorded in the local to the client time zone. This information is only available at the dealer, it is uploaded as the message metadata alongside with the encrypted request or report and not forwarded to the broker. As expected, there's a no-

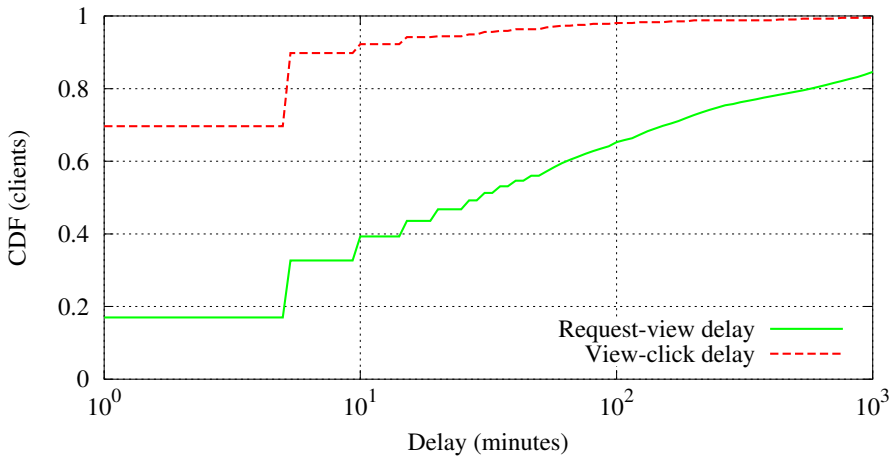


Figure 8.4: CDFs of delays between ad requests and ad views, and between ad views and ad clicks

ticeable trough corresponding to nightly hours – all distributions exhibit a clear diurnal pattern.

Figure 8.4 plots the distributions of delays between an ad request and the first ad view for the generated channel, and between and ad view and the corresponding ad click. The former is an inherent attribute of the private-by-design architecture and depends on the delays between system components. Additionally, the request-view delay is affected by the channel selection and ad placement policy at the client and is subject to the availability of the adboxes. Overall, our prototype was able to generate, deliver and display an ad within an hour of establishing a new interest for 60% of all channels. With the most aggressive configuration parameters (channel selection = 'most recent' and ad placement = 'everywhere') the 60th percentile of the delay is below 30 minutes. We believe this to a large extent comes from the time it takes for an adbox to show up.

Additionally, using PDDP we measured the delay between sending an ad request and receiving ads from the generated channel (this information is only

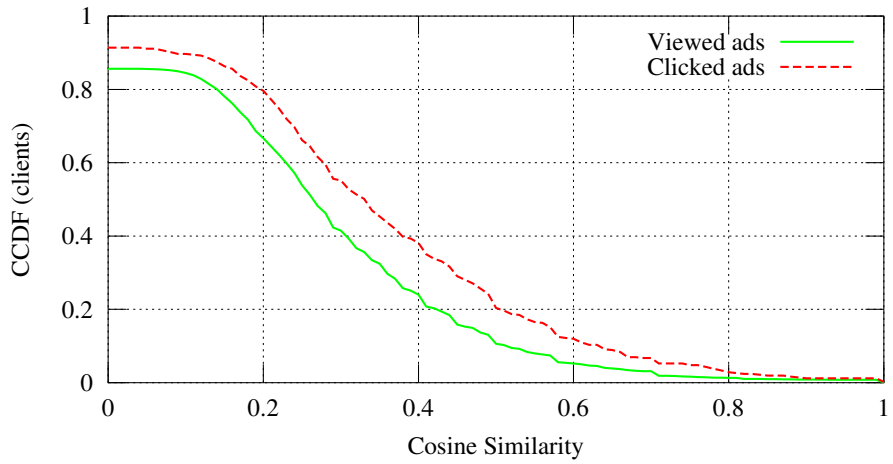


Figure 8.5: CCDFs of cosine similarity between terms that appear in ad content and in the corresponding ad request for clicked and viewed ads

recorded locally at the client). This delay for all responded clients was <10 minutes. There are several contributing factors to the request-response delay: length of the upload cycle at the dealer, polling interval at the client and the ad generation latency. While the first two are configurable, the latter is implementation-specific. In our experiments, the observed average ad generation latency was below 2 seconds (with 99th percentile of 7 second).

While, as Figure 8.4 shows, the majority of the clicks happen within minutes after an ad view (the 90th percentile is less than 10 minutes), we do not have enough data points to establish the relationship between CTR and time elapsed since identifying a new user interest. Nonetheless, we found that CTR is affected by the number of times an ad was shown (so-called, 'opportunities to see' or *ots*), with the first, the second and the third ots accounting for 70%, 14% and 10% of all clicks respectively.

To evaluate the relevance of Privad ads to user interests, we compute the cosine similarity using terms from the generated ad content and from the cor-

responding ad request for all clicked and viewed ads. With even such a simple metric, which does not take into account semantic relevance, it is clear that clicked ads tend to better match the original request than non-clicked ads. As Figure 8.5 shows, almost 40% of the clicked ads have 0.4 similarity coefficient with ad requests, whereas for viewed ads this number is slightly more than 20%. We manually inspected clicked ads with similarity lower than 0.1 and found that the majority of corresponding requests contain terms from non-English languages and do not appear in the actual ad content.

Overall, we find the achieved CTR to be encouraging given the fact that we fully relied on shopping engines to match ad requests to relevant products and that our ad content was produced automatically from resulting products (which, despite our best effort, sometimes did not look as intelligible and appealing as handcrafted ads). Naturally, we could have seen higher CTR by rendering our ads only in top positions. Apart from this, we believe that Privad CTR can be increased by improving targeting heuristics (e.g., reducing noise in search-based targeting), investing in better request matching algorithms and serving advertising content designed by hand, not auto-generated.

CHAPTER 9

COLLECTING DATA WITH PDDP

As opposed to the aggregate performance stats maintained by Privad using view and click reports, PDDP provides a privacy preserving mechanism to collect per-user stats and perform user-centric analysis. In this chapter, we describe our experience exploring the extent to which a differentially private data collection system can be used to understand what is going on behind the scenes in the private-by-design ad deployment.

Towards this end, we start out by building confidence in the differentially private results by collecting attributes that characterize the Privad's user population and comparing them with the data available at the server-side. We then exercise PDDP functionality to get better visibility into the client-side and understand the reasons for the observed view and click rates. We also analyze advertising performance for Google ads and compare it with Privad's. We examine the difference between search and display ads in terms of the before- and after-click user behavior. Additionally, we attempt to discover the behavioral differences between clicking and non-clicking users. Finally, we look at the privacy deficits accumulated as a result of our analysis and study the privacy implications for the end users.

In general, our primary goal in evaluating PDDP was not to arrive at any definite conclusion per se (our user population is not sufficiently large for this purpose), but to demonstrate the types of analysis made possible with PDDP.

9.1 Aggregate User Characteristics

We start our PDDP data collection with two simple queries, one retrieving the user timezone, the other the user geographical region. Timezone information is available to addons as a part of the Firefox API. To obtain the geographical data the clients call the Maxmind GeoIP API. Both values are updated whenever the browser restarts and are stored in the client's SQLite database. The respective queries are simple one-line select statements with buckets enumerating all possible timezones in one case, and the top 20 most represented countries (in terms of the volume of IPs that appear in the dealer's logs) in the other. Both queries were active throughout a 24-hour time interval (which means that their result sets contain answers from all clients who remained online long enough to receive and execute the query, and encrypt and submit the answer). In total, we collected 4852 answers for the timezone query with 588 random coins added to each bucket,[1] which corresponds to a standard deviation of 12.12. For the geographical region we obtained 4604 answers with 585 per-bucket coins (with stdev = 12.09).

Figure 9.1 shows the distribution of users over timezones collected using PDDP, together with the distribution of messages received by the dealer during the same 24-hour time span constructed using the local timestamp from the message meta-info field. The two clusters in the histograms center around

[1]For all PDDP queries in our analysis we use $\epsilon = 1$.

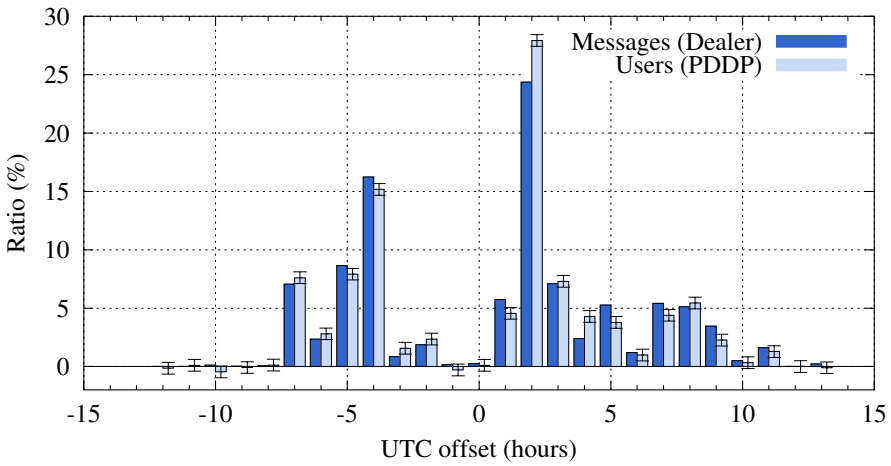


Figure 9.1: Timezone distribution. Error bars correspond to two standard deviations = 0.5% (95% confidence interval).

the EDT (UTC-04:00) and CEST (UTC+02:00) timezones, which correspond to American and European user populations. While both distributions generally have the same form, a few individual histogram bars diverge by up to 4%. One possible explanation is the fact that the volume of generated messages varies from user to user, with an adblocked user generating only a fraction of messages generated by a non-adblocked user (no view and click messages). Also, the number of messages depends on the browsing behavior (e.g., performing searches vs. following links) and on the duration of online activities.

Figure 9.2 plots two distributions of Privad users over a list of countries. One is based on the noisy answers from the second PDDP query, the other is computed using 4607 IPs extracted from the dealer's logs from the same 24-hour period. The tallest bar on the figure corresponds to the US users, and the more sizable European population is spread over a number of countries including Germany, Russia, Great Britain, France and others. Almost all values computed from back-end data lie within the 95% confidence interval of respective PDDP values, with the exception of the Spain and NA ratios. The reason for this mi-

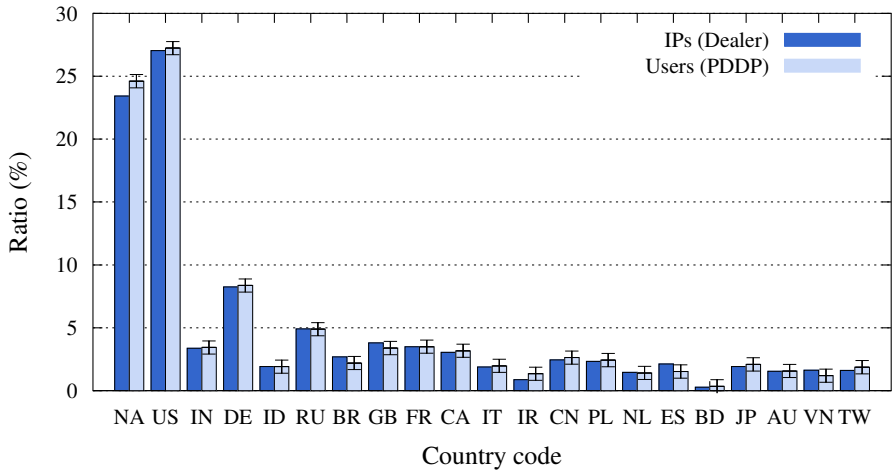


Figure 9.2: Geographical distribution. Error bars correspond to two standard deviations = 0.5% (95% confidence interval). NA represents countries not included in the top-20 list.

nor mismatch is likely to be that local GeoIP info is captured as soon as the browser starts and is not updated until the next browser restart, therefore it can be somewhat stale by the time a client received the PDDP query. For example, if a user enabled a proxy or joined a different network, the IP address recorded in the dealer's log can be different from the one used to retrieve local GeoIP info. Overall, we find that the back-end data confirms trends discovered using the differentially private mechanism, which serves as a practical validation of PDDP results.

We use the rest of this section to describe our experience in exercising the PDDP functionality to the widest extent possible in order to learn everything we could about our deployment.

To build a better picture of Privad users, we used PDDP to query the operating system installed on the user machine (also available as a part of Firefox API). We found that among the 4428 clients who responded to the query, 64.2% run Windows, 21.6% use OSX and 14.4% have a Linux installed. Clearly, with the

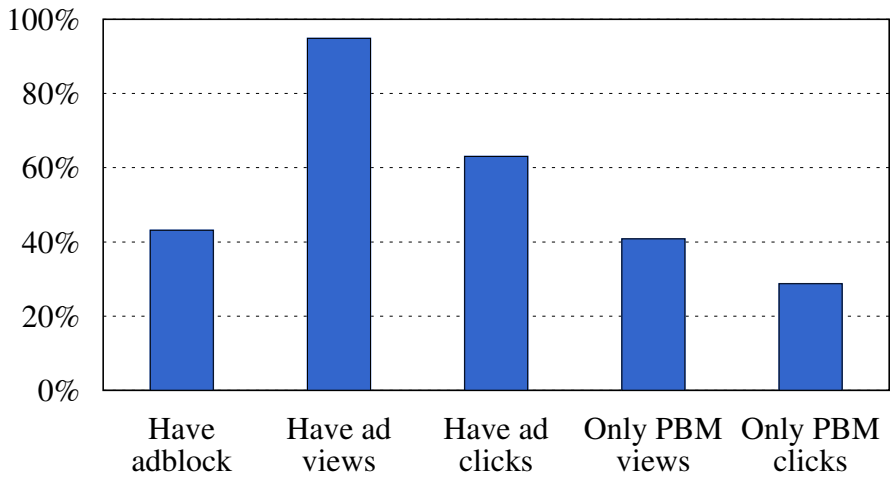


Figure 9.3: User attributes collected with PDDP. 95% confidence interval is ±0.6%. PBM stands for private browsing mode.

ratio of Linux users significantly higher than in the general population (1.73% according to [59]), this sample is representative of a set of technically savvy power users.

Finally, we executed a query to find the percentages of adblocked users as well as users with views and clicks for all types of ads (both Google and Privad). The query was active for 96 hours and accumulated a total of 5909 answers. Among the users who submitted their answers ca. 709 spent less than two weeks with the system and their values were not included in the query results. Figure 9.3 shows the distribution of the remaining user answers over queried attributes. Consistent with the previous observation about the tech-savvy user population, we found the ratio of ad-blockers among Privad users to be twice the rate for Firefox users reported by [20]. Surprisingly, despite a large number of adblocked users, the vast majority still receive ad views (mainly because popular ad-blocking addons consider Google Search ads to be "acceptable" [2] and by default do not remove them; by contrast, *no* Privad ads are displayed if our client detects an adblock). Moreover, more than 63% of users in

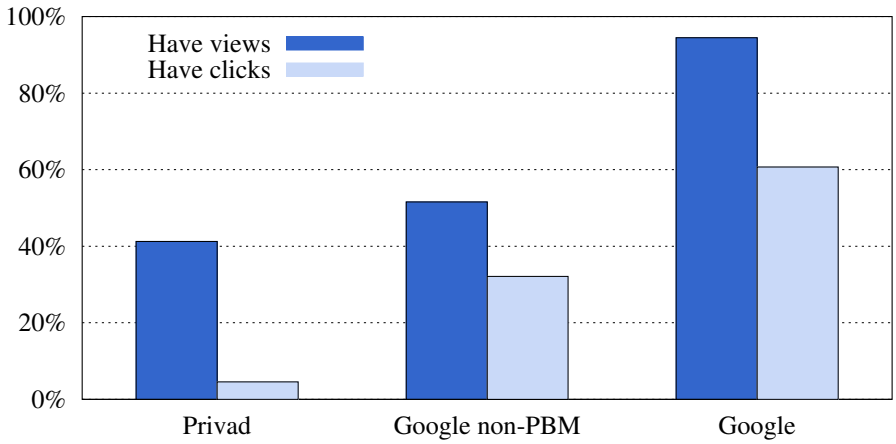


Figure 9.4: Ratios of users with views and clicks. Timespan = 96 hours, total answers = 5584, 95% confidence interval = ±0.7%.

the sample have ad clicks (for users who block ads this ratio is only marginally smaller – 60%). Additionally, 40% and 28% of users have views and clicks registered only during private browsing mode (PBM). This suggests that more than a third of Privad users have PBM enabled most of time, during which Privad client will neither request nor display any ads.

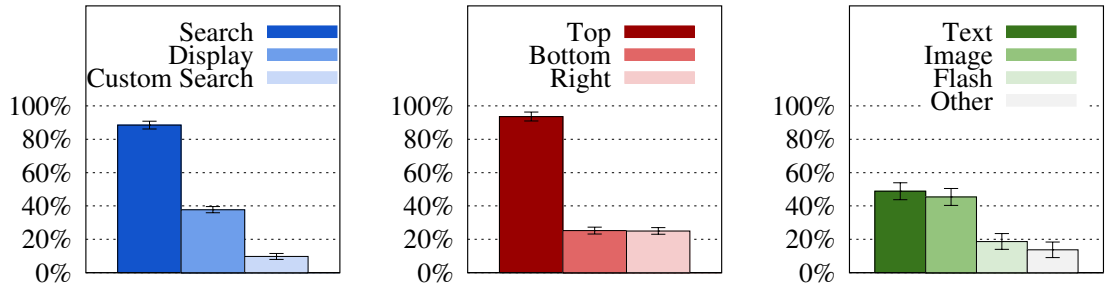
Overall, our PDDP-enabled analysis reveals a technically advanced user base, where a large fraction have ad-blocking addons and browse in private mode. The outcome is far from unexpected, given our "bundling" deployment mechanism. On top of that, the users who opt into a study of private-by-design advertising are likely to be aware of the privacy concerns related to online advertising and to use all means available to minimize their privacy exposure on the web.

9.2 Comparing Privad and Google Performance

In this section, we present the results of PDDP analysis performed to compare and contrast Privad and Google ads performance. Figure 9.4 plots the ratios of users with views and clicks for both Privad and Google ads. It shows that less than 5% of sampled users have Privad clicks, whereas more than 30% have non-PBM Google clicks. We did not store any details about events that occurred in PBM apart from the type of event (i.e., 'view', 'click', 'search', etc). But since no Privad ads are displayed in PBM, all ad-related PBM events are attributed to Google. In total, we found that more than 60% of all queried users have registered Google clicks. Due to lack of information about events in PBM and to be apples-to-apples with Privad, unless otherwise stated, we exclude PBM views and clicks from further analysis.

To learn which Google ads are most popular, we break down users with clicks into groups according to the type of the clicked ad. As Figure 9.5a shows, among the approximately 1387 users with Google clicks in this sample, the vast majority registered a click on a search ad. Display with almost 40% of all clickers is runner-up. Figure 9.5b further breaks down search ads according to location, where top ads (displayed above organic search results) take the lead with more than 90% of 1227 users with search ad clicks. Finally, among display ads (Figure 9.5c), text and image have roughly equal shares each with 40% of users with display clicks (ca. 524 users in the sample).

To estimate the distribution of per-user Google click-through rates, we issued a query computing the ratio of views to clicks that occurred in October 2013 for all Google ads (including ads displayed in PBM). Since every bucket in



(a) All Google ad. (b) Search ads (c) Display ads

Figure 9.5: Breakdown of users with at least one Google click by type of the clicked ad. Timespan = 24 hours, total answers = 4624.

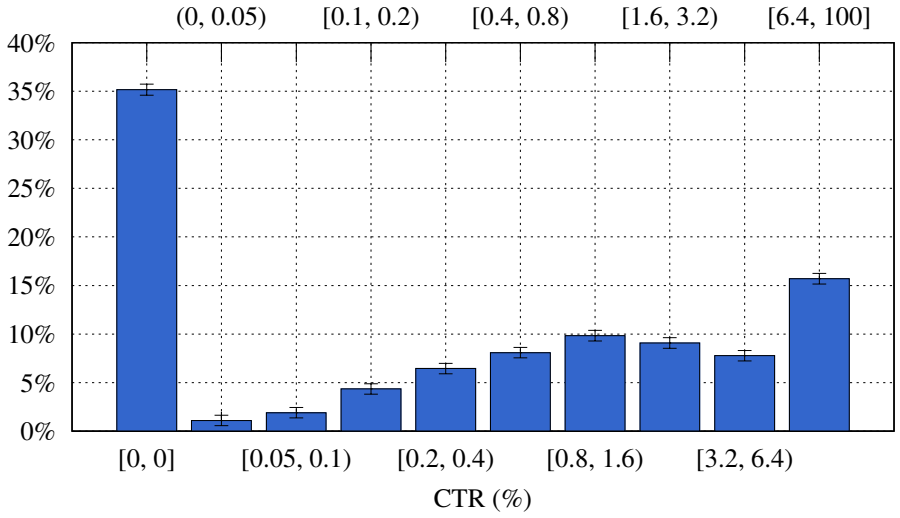


Figure 9.6: Distribution of per-user Google click-through rates. Timespan = 96 hours, total answers = 5392. Error bars denote 95% confidence interval. Labels along X-axis represent corresponding bucket ranges.

a PDDP query incurs a penalty in terms of the added privacy deficit, we generally strove to use as few buckets as possible. In particular, in order to cover the whole range of possible CTR values in this query we used exponentially increasing bucket sizes (with every bucket covering twice the range of its precursor). Among 5392 user who responded to the query, 865 did not have any views, CTR values for the remaining ca. 4527 users are distributed as shown in

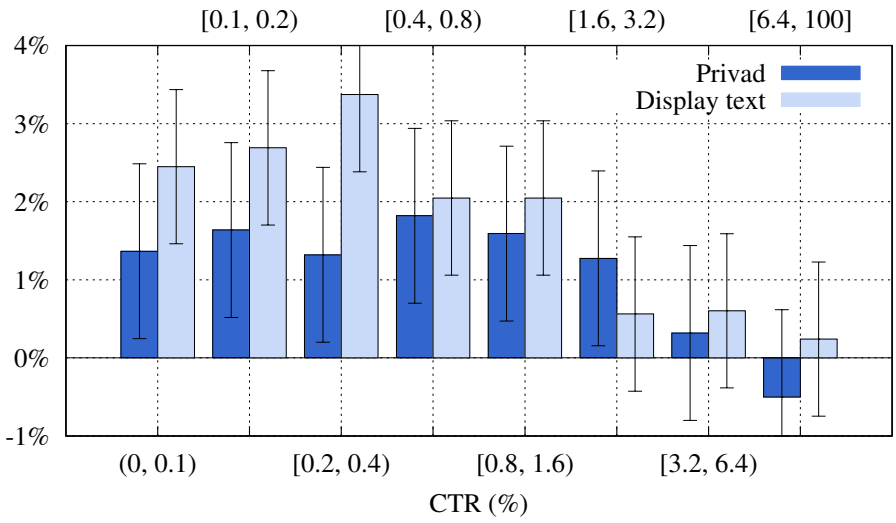


Figure 9.7: Distributions of per-user click-through rates for Privad and Display text ads. Timespan = 96 hours, total answers = 6317. Zero-CTR ratios ($90 \pm 1.5\%$ for Privad, $85 \pm 1.3\%$ for display text ads) not shown.

Figure 9.6. The first histogram on the figure represents the ratio of users with no clicks, while the last – the ratio of users with CTR values $\geq 6.4\%$. Using the lower endpoint of a bucket range as a conservative estimate of the CTR value for a user within the bucket, we compute a lower bound on the average Google CTR across all users – 1.53%.

Privad is an inherently *asynchronous* system designed for behavioral advertising, not search advertising. The prototype we built generates and displays only contextual ads. Therefore, it is most reasonable to compare performance of Privad ads and text ads on the Google Display Network (which we refer to as *display text ads*). To do so, we executed a query computing per-user CTR values for both types of ads. The results of this query are presented in Figure 9.7, excluding the ratios corresponding to zero-CTR. While dominated by differentially private noise, both CTR are generally quite similar.

To verify this observation, we used PDDP to compare the Privad and display text CTR values of each individual user. We collected 4844 answers with 588 coins added to each bucket (stdev = 12.1), among which 2994 had no Privad or display text views, and 1460 had no clicks. Among the remaining 391 users, we found that in 210 cases the display text CTR was higher, and in 181 the Privad CTR was higher.

In addition to detecting views and clicks, Privad clients use a number of heuristics to identify conversion (i.e., purchases made after a click). Towards this end, clients tag all pages visited after an ad click with the ad id and try to detect online payments (e.g., credit cards numbers in post request parameters that pass through Luhn's validation) issued on pages tagged with a valid ad id. To compare how Privad, with 4 actual conversions, fares against Google, we executed a PDDP query counting the number of detected Google conversions. From 6022 users who responded, 107 users made a purchase following a click on a Google ad (stdev = 12.3), 23 made a conversion while in PBM and 29 had more than one conversion. However, the majority of conversions (ca. 92) were generated by search ads, with display ads accounting for only 7 conversions (well below the noise level).

In general, using PDDP we found that both in terms of the click-though rates and conversions Privad ads perform on a par with text ads on Google Display Network. While recommendation systems and ad targeting are well studied in as long as they are performed in the cloud, it still remains a challenging and open question how to do targeting on the client using only the local profile information. The preliminary results reported here suggest that even with a few simple heuristic for targeting from the localhost Privad is as effective as the ad

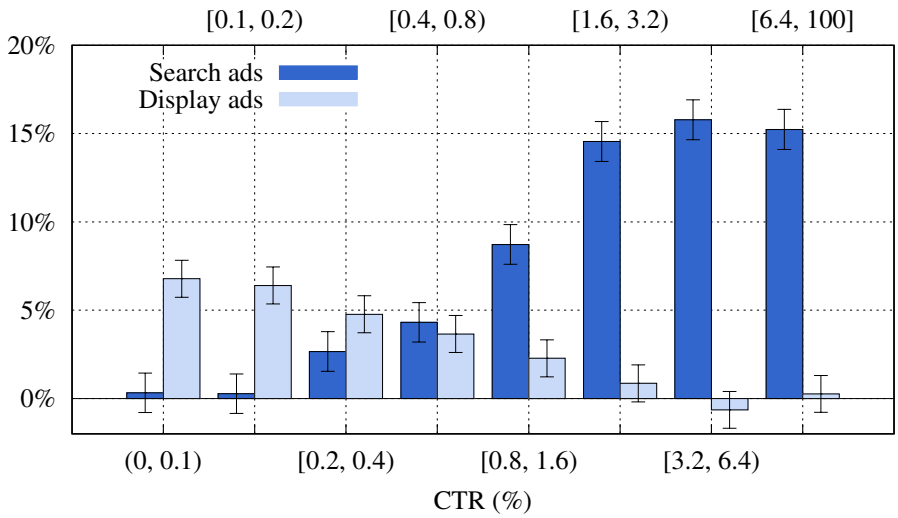


Figure 9.8: Distributions of per-user click-through rates for Search and Display Google ads (all subtypes). Timespan = 72 hours, total answers = 5272. Zero-CTR ratios (40 ± 1.2% for Search, 74.4 ± 1.3% for Display) not shown.

network, which has a global view of the user profiles and employs complex machine learning algorithms.

9.3 Comparing Search and Display Performance

As mentioned in the previous section, our PDDP analysis reveals a difference in the number of conversions attributed to search and display ads. The dissimilarity between these ad types is even more prominent when we compare the distributions of per-user click-through-rates (Figure 9.8). More than half of roughly 2200 users with search ad views have CTRs $\geq 0.8\%$, whereas display ad CTRs for almost all users do not exceed this value. Also, all search clicks were generated by text ads, but half of the display clicks comes from text ads and half from image ads.

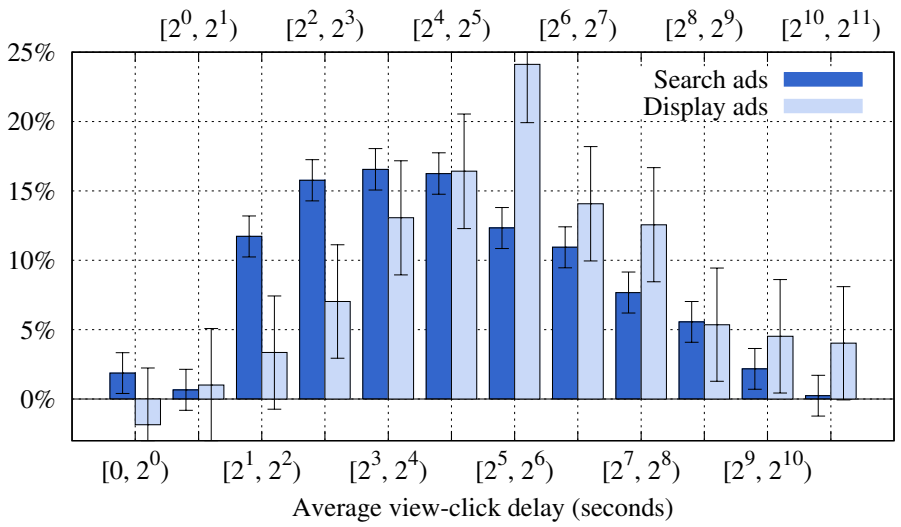


Figure 9.9: Distributions of per-user average view-click delay for Search and Display text ads. Timespan = 48 hours, total answers = 5239.

After observing a stark difference in performance between search and display ads, one of the remaining questions was whether these two ad types differ in terms of the before- and after-click user behavior. In other words, we want to find out how long users linger before clicking on an ad and whether they remain interested in and engaged with the landing page content a long time or leave soon after the click. Our primary goal here was to exercise the PDDP functionality to the widest extent possible without making any significant claims regarding observed results. Therefore, we compare search and display focusing only on text ads. In order to increase size of the population with display text clicks, we include Privad clicks in this category, since both display text and Privad ads are visually similar and share the same performance characteristics.

Figure 9.9 plots the distribution of the average per-user delay between an ad view and the subsequent click on the ad. The mean delay for display ads is around 30 seconds to 1 minute, while for search ads it is between 8 and 30 seconds. In part, this difference could be explained by the fact that we register

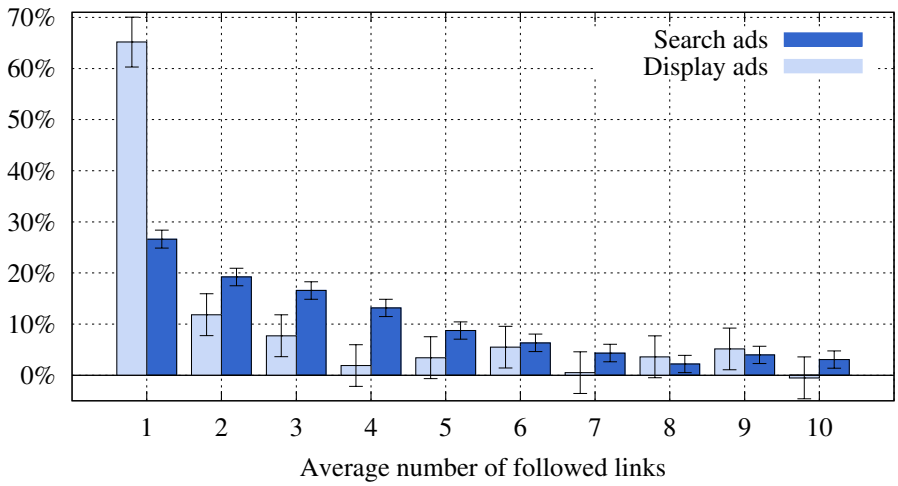


Figure 9.10: Distributions of per-user average click-chain length for Search and Display text ads. Timespan = 48 hours, total answers = 4895.

a view at the time of the page load, and not when an ad is within the visible area of the browser window. And while top search ads appear immediately on the screen, oftentimes a user has to scroll down to see a number of display ads hidden outside of the visible area.

Another metric we use to analyze post-click behaviour is the length of the "click-chain" – the number of pages visited by following links on the advertiser's webpage. The distributions of the average click-chain length are shown on Figure 9.10. On average, the majority of users visit only the landing page of a display text ad (click-chain length = 1) and rarely follow 1-2 additional links (counts corresponding to buckets with more than 3 visited pages are dominated by differentially private noise). The same distribution for search ads is more evenly spread with bucket counts corresponding up to 7 visited pages well above the noise level.

Finally, Figure 9.11 presents the distributions of time spent actively browsing advertiser's content for both ad types. Again, the figure clearly shows a shift

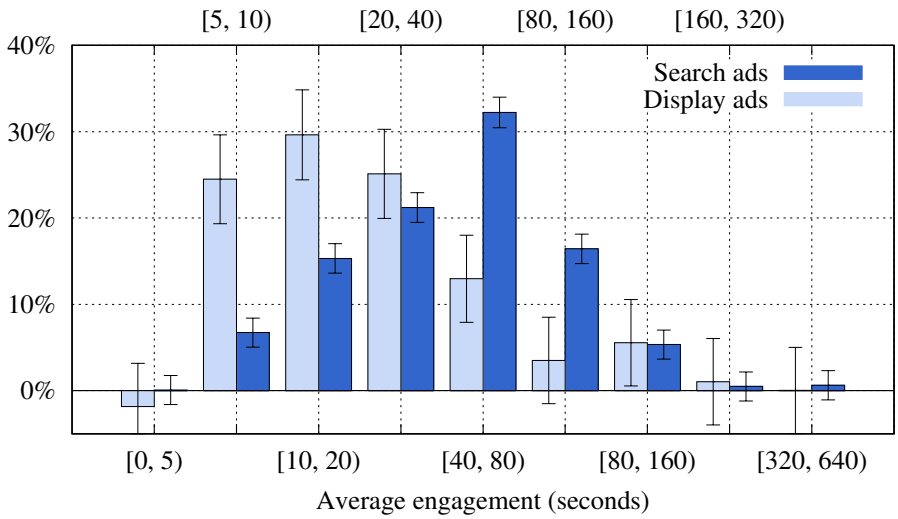


Figure 9.11: Distributions of per-user average engagement for Search and Display text ads. Timespan = 48 hours, total answers = 5059.

between distributions with the average engagement reaching a peak at 10 to 20 seconds for display ads and 40 to 80 seconds for search ads.

9.4 Clickers vs. Non-clickers

In addition to comparing performance characteristics for different types of ads, PDDP also enabled us to analyze online behavior of different user groups. Specifically, we attempted to discover correlations between clicking and a number of other online activities. For example, using PDDP we compared users with zero-clicks (aka non-clickers) and users with two or more clicks (aka clickers) in terms of the frequency of online shopping. For each user we computed the number of items placed in a shopping cart (i.e., shopping events) divided by the total active browsing time. Figure 9.12 plots the distributions of this ratio for both user groups. On average, as the figure shows, clickers tend to shop online more often than non-clickers. Unfortunately, we did not find any strong

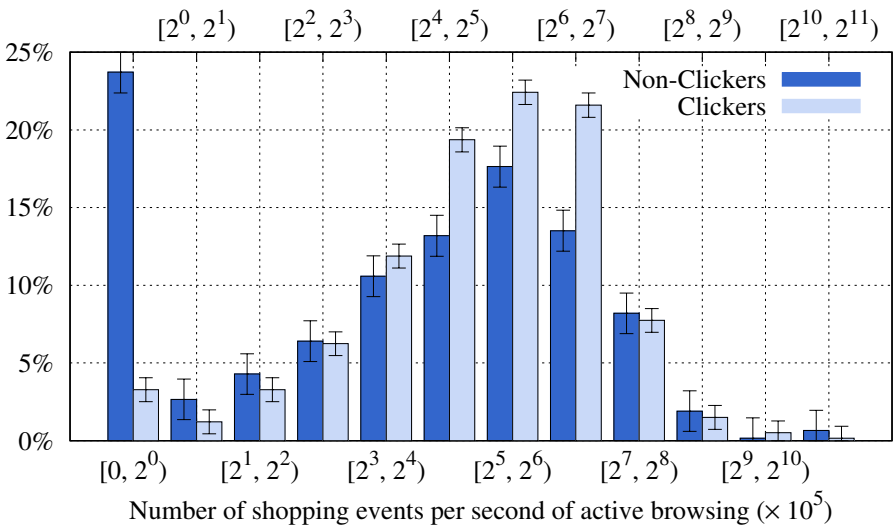


Figure 9.12: Distributions of the number of shopping events per second of active browsing time (scaled by a factor of 10^5) among clickers and non-clickers. Timespan = 96 hours, total answers = 5911.

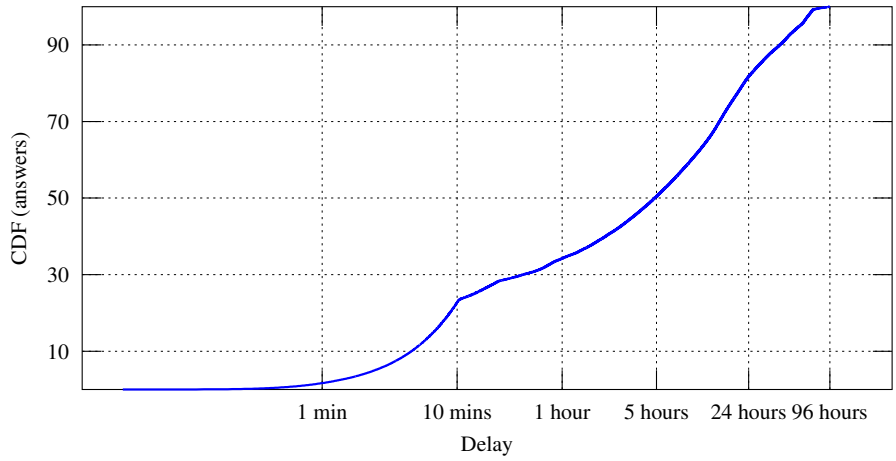


Figure 9.13: Distribution of delay for collected PDDP answers

correlations that would allow to classify a user as a clicker or non-clicker based only on the locally available information about user's online activities.

9.5 Analysis

Overall, we executed 159 PDDP queries, collecting 790017 answers from 9395 unique clients. Figure 9.13 plots the CDF of the fraction of answers collected by the dealer within a certain period after issuing a PDDP query. The figure shows that a half of all answers were received within 5 hours after launching a query. The majority of these answers come from the European user population located not farther than a few timezones away from the dealer. Not surprising, to reach more users around the globe we had to specify query lifetime of at least 24 hours.

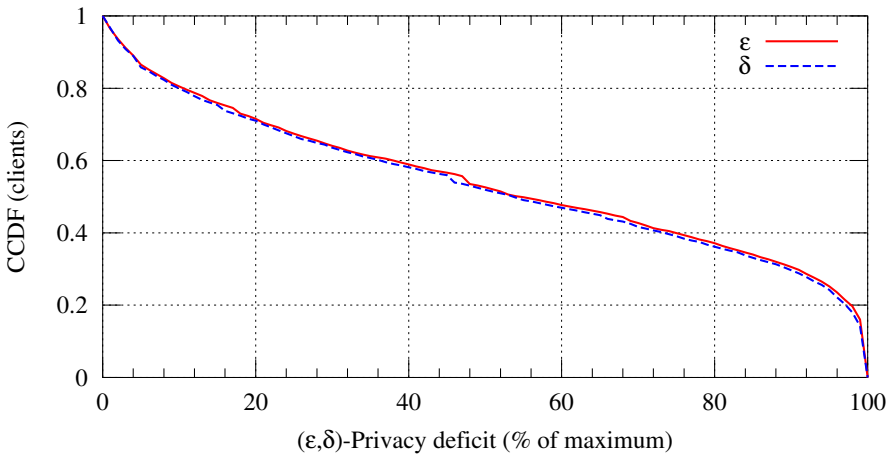
For all queries we used $\epsilon = 1$ and added a number of coins to satisfy the $\delta < 1/c$ requirement [19]. As already mentioned, we treat individual buckets as separate queries and, therefore, compute the (ϵ, δ)-privacy cost of a query as $(1, 1/c)\times$ *number of buckets*. The cumulative privacy deficit for each client is recorded at the dealer. This dataset shows that the maximum per-client privacy cost of our PDDP analysis is $(\epsilon = 3006, \delta = 0.6)$. Moreover, as shown in Figure 9.14a 20% of the clients who participated in the analysis accumulated a deficit of more than 97% of the maximum. One way to understand this amount of privacy deficit is to estimate how many user attributes it will allow an attacker to learn under a worst-case scenario assumption.

Isolation attack (averaging out the noise). Let's assume that an attacker can isolate a single client (for instance, by constructing a query in such a way as to single out one user) and repeatedly pose the same query to this client (or alternatively a single query with a large number of identical buckets). If the system adds at least n coins to an answer, the noise generated in a single trial is

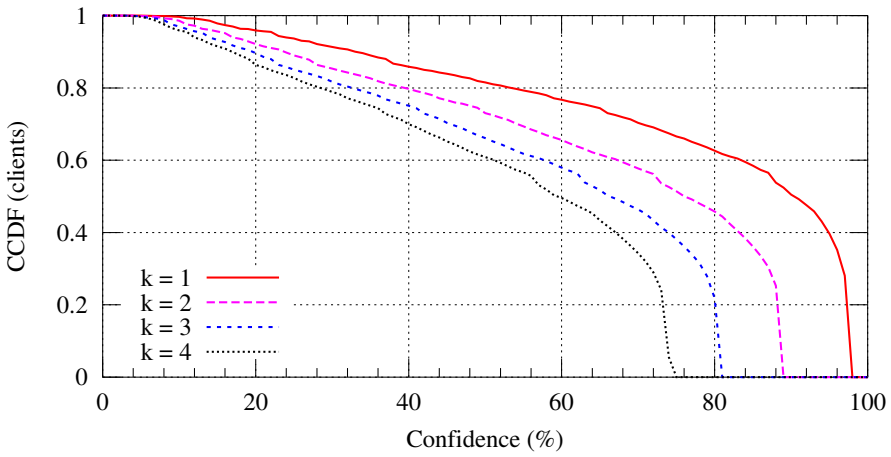
$\sum_i^n x_i - n/2$, where x is value of an individual coin (0 or 1). The expected value of the noise in a single trial is 0 (noise is distributed binomially with success probability $p = 0.5$). The average noise value after t trials is $\frac{1}{t}\sum_k^t(\sum_i^n x_i - \frac{n}{2}) = \frac{1}{t}(\sum_i^{nt} x_i - \frac{nt}{2})$, which according to the central limit theorem follows approximately normal distribution $\mathcal{N}(0, n/4t)$. 95% of the values of this distribution lie within two standard deviations of the mean: $[-\sqrt{n/t}, \sqrt{n/t}]$. The noise is effectively cancelled out when the width of this interval is < 1. In other words, if after t trials an attacker observes average PDDP value from $(-0.5, 0.5)$ interval, it infers that the true user value is 0 with 95% confidence (similarly a value from (0.5, 1.5) interval corresponds to the user value of 1). Solving $\sqrt{n/t} < 0.5$ for t, gives that if $t > 4n$ the noise is cancelled out with 95% confidence (for 99% confidence t $> 9n$).

The maximum (ϵ, δ)-privacy deficit of (3006, 0.6) in our dataset allows an attacker a single query with $t = 3006$ identical buckets and $n = 590$ coins added to each bucket. This enables the attacker to learn a single sensitive user attribute with confidence >97% (alternatively, using two queries with 1503 buckets the attacker can learn two attributes with confidence >89%). Figure 9.14b plots the attacker's confidence across all users, based on the number of answers produced by each user. This shows that, according to the differentially private model, an attacker could have predicted a single attribute for 40% of the users with 95% confidence (or two attributes with 83%).

We cannot, however, conceive of any query that would have allowed us to learn what Figure 9.14b suggests. Imagine that we had enough auxiliary information about *one* of the clients to formulate a query that isolated that client from all the others. Then using a malicious 3006-bucket query, we could have



(a) Distribution of accumulated per-client privacy deficit measured by the dealer. Maximum ϵ-deficit = 3006, maximum δ-deficit = 0.6



(b) CCDFs of the attacker's confidence of discovering k user attributes given each user's privacy deficit (theoretical worst-case)

Figure 9.14: Accumulated privacy deficit and its practical implications

learned one thing about our victim. But we would have learned nothing about the other clients that answered the query, even though theoretically those clients experienced privacy loss.

In other words, not only does the differential privacy worst-case model give a poor reflection of the actual privacy loss experienced by our client population (i.e. none!), it is a poor reflection even for the best attack we can conceive. A conservative interpretation of differential privacy would suggest that we should

have made fewer queries than we made. In practice, this would have unnecessarily hampered the utility of our system. Somehow this gap between theory and practice needs to be reduced. For example, by introducing a number of additional privacy mechanisms which range from simple tricks like limiting the number of buckets in a query [10], to running taint analysis to ensure that the same attribute is not used to answer repeated queries. While not provably differentially private, these practical mechanisms will nevertheless raise the bar for attackers.

CHAPTER 10

SUMMARY AND FUTURE DIRECTIONS

In the first part of this thesis, we addressed the challenge of designing an online advertising auction for a private-by-design advertising system that leverages user profile information while keeping the user profile private. We broadly explored the design space, proposing three types of auctions, and analyzing their properties with respect to privacy, auction quality and vulnerability to attack. Overall, we found that two of the systems, Rank-at-Client (RaC) and Rank-at-3rd-Party (Ra3), are very acceptable designs. RaC is simpler and more efficient, but has the drawback that information about ad quality and bid is leaked. On the other hand, this is information that can today be determined by placing ads and monitoring the resulting ranking. Finally, noting that our auction designs suffer delays that cause out-of-date bid information to be used in rankings, we use Bing advertising system auction trace to determine the effect of these delays. We find the effect to be very minimal, and so conclude that our auction designs are viable.

In the second part, we described our experience and challenges involved in building, deploying and operating a private-by-design ad system. Much of this work is empirical in nature, and as such is full of experimental warts and idiosyncrasies (mostly acknowledged), but in a sense that was the point. Overall, we believe that the process we went through to see our prototype deployed and used by thousands of users is as much a contribution of this work as the results we obtained. This experiment provided us with ample evidence and helped answer several key questions such as: is the private-by-design technology a non-starter, what can a researcher do to evaluate a private-by-design system,

short of an actual start-up, and what compromises are made in the process. We learned several lessons from this experiment.

First, Search ads, which reflect user interests in real-time, clearly perform better than Display ads. Therefore, technologies like Privad, which delay ad delivery, face a serious limitation. On the other hand, compared with Display text ads, Privad performed surprisingly well. A number of additional improvements can be made to achieve even higher click-through rates.

Second, using PDDP analytics we learned that the population of users who opted into the study was biased towards technically advanced power users, many of whom have ad blocking software, browse in private mode and tend to rarely click on ads. Therefore, we believe that the observed performance is an overly conservative estimate of the click-through-rates in general.

Third, when used with Privad's threat model, which assumes an honest-but-curious adversary, differential privacy produces an excessively pessimistic estimate of privacy loss. In reality, we came nowhere near learning any attributes about any specific individual. Additionally, an expected, but still negative result of this study is that, without additional measures to raise the attack bar, differential privacy is inadequate for long-running analytics.

Unfortunately, our experiments do not allow us to address one important question: what is required to incentivize adoption of the private-by-design technologies. Nonetheless, since the private-by-design model was proposed in response to public concerns over ever increasing online tracking, we believe that the most realistic path towards wide-spread adoption of this model goes through establishing and enforcing sensible privacy policies on the Inter-

net. These policies must create sufficient regulatory pressure on the advertising industry, so that it will abandon its current privacy invasive practices and start looking for alternative technologies. In this sense, our experiments with private-by-design model serve as an argument in the "tracking versus targeting" debate and show that a viable alternative indeed exists.

There are still a number of open challenges in the space of private-by-design advertising. These include developing better algorithms for profiling and targeting from the localhost, protecting after-click user privacy (i.e., privacy from advertisers), and reducing inherent delays in the architecture in order to serve real-time search ads. A fully functional auction component requires a mechanism to compute user score. Such a mechanism is yet to be designed and evaluated in terms of the effectiveness of predicting click-through rates. A measurement study of ads served by a major search engine could help determine to what extent advertisers can reverse-engineer each otherś bids in today's systems. This will quantify how much advertiser privacy loss is incurred by the Rank-at-Client scheme.

Each of the private-by-design advertising schemes so far proposed makes the tacit assumption that there is only a single broker, and a single profiler operating at each client. It is unclear what happens if there are multiple brokers with competing profilers in each client. In particular, the profilers should be able to dynamically compete for ad boxes in real time, thus adding a new element to the auction that is not unlike the way ad exchanges operate today. Moreover, these clients may also need to compete with existing tracking advertising systems in real-time auctions run by existing ad exchanges.

The last step in Privad's evolution is to build an auction component and deploy the system within an ad exchange. However, we do not believe this is feasible in a purely research setting. Rather, a commercial deployment is needed.

BIBLIOGRAPHY

[1] Alexa - The Web Information Company. http://www.alexa.com/.

[2] Allowing acceptable ads in Adblock Plus. https://adblockplus.org/en/acceptable-ads.

[3] Apache Thrift. http://thrift.apache.org/.

[4] Google Docs - Viewer. https://docs.google.com/viewer.

[5] Google Docs Viewer (PDF, DOCX, PPTX, XLSX, etc...). https://addons.mozilla.org/en-US/firefox/addon/google-docs-viewer-pdf-doc-/.

[6] iFamebook (Facebook Profile Visitors). https://addons.mozilla.org/en-us/firefox/addon/ifamebook/.

[7] InvisibleHand: Home. http://www.getinvisiblehand.com/.

[8] Redesigned Text Ads on the Google Display Network. http://adsense.blogspot.com/2013/09/redesigned-text-ads-on-the-Google-Display-Network.html.

[9] M. Abe and K. Suzuki. M+ 1-st Price Auction Using Homomorphic Encryption. In *Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, pages 115–124. Springer-Verlag, 2002.

[10] Istemi Ekin Akkus, Ruichuan Chen, Michaela Hardt, Paul Francis, and Johannes Gehrke. Non-tracking Web Analytics. In *ACM Conference on Computer and Communications Security*, 2012.

[11] Sebastian Angel and Michael Walfish. Verifiable auctions for online ad exchanges. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 195–206. ACM, 2013.

[12] O. Baudron and J. Stern. Non-interactive Private Auctions. In *Proceedings of the 5th International Conference on Financial Cryptography*, pages 364–378. Springer-Verlag, 2002.

[13] Peter Bogetoft, Dan Lund Christensen, Ivan Damgrd, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes live. In *Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

[14] F. Brandt. Fully private auctions in a constant number of rounds. In *Proceedings of the 7th international conference on Financial cryptography*, pages 223–238. Springer-Verlag, 2003.

[15] F. Brandt and T. Sandholm. Efficient privacy-preserving protocols for multi-unit auctions. In *Proceedings of the 9th international conference on Financial cryptography and Data Security*, pages 298–312. Springer-Verlag, 2005.

[16] Felix Brandt. How to obtain full privacy in auctions. *International Journal of Information Security*, 5(4):201–216, 2006.

[17] Benjamin S Lerner Brian Burg and Herman Venter Wolfram Schulte. C3: An experimental, extensible, reconfigurable platform for html-based applications. In *2nd USENIX Conference on Web Application Development*, page 61, 2011.

[18] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 120–127. ACM, 1999.

[19] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johannes Gehrke. Towards Statistical Queries over Distributed Private User Data. In *NSDI*, 2012.

[20] ClarityRay. About Ad-Blocking. `http://clarityray.com`.

[21] Stephanie Clifford. Instant Ads Set the Pace on the Web. *The New York Times*, March 2010. `http://tinyurl.com/yl8dt29`.

[22] DoubleClick. DART for Advertisers. `http://www.doubleclick.com/products/dfa/index.aspx`, 2009.

[23] Cynthia Dwork. Differential privacy. In *Proceedings of the 33rd international conference on Automata, Languages and Programming - Volume Part II*, ICALP'06, pages 1–12. Springer-Verlag, 2006.

[24] Cynthia Dwork. Differential privacy: a survey of results. In *Proceedings of the 5th international conference on Theory and applications of models of computation*, TAMC'08, pages 1–19. Springer-Verlag, 2008.

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

[26] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *Information Theory, IEEE Transactions on*, 31(4):469–472, 2002.

[27] J. Feng, H.K. Bhargava, and D.M. Pennock. Implementing sponsored search in web search engines: Computational evaluation of alternative mechanisms. *INFORMS Journal on Computing*, 19(1):137, 2007.

[28] Matthew K Franklin and Michael K Reiter. The design and implementation of a secure auction service. *Software Engineering, IEEE Transactions on*, 22(5):302–312, 1996.

[29] Matthew Fredrikson and Benjamin Livshits. Repriv: Re-imagining content personalization and in-browser privacy. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 131–146. IEEE, 2011.

[30] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[31] Google, Inc. AdWords: Advertise Your Business on Google. `http://adwords.google.com/`.

[32] Saikat Guha, Bin Cheng, and Paul Francis. Privad: Practical Privacy in Online Advertising. In *Proceedings of the 8th Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, Mar 2011.

[33] Saikat Guha, Bin Cheng, Alexey Reznichenko, Hamed Haddadi, and Paul 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`.

[34] Saikat Guha, Alexey Reznichenko, Kevin Tang, Hamed Haddadi, and Paul Francis. Serving Ads from localhost for Performance, Privacy, and Profit.

[35] Hamed Haddadi. Fighting online click-fraud using bluff ads. *ACM SIGCOMM Computer Communication Review*, 40(2):21–25, 2010.

[36] Hamed Haddadi, Pan Hui, and Ian Brown. Mobiad: private and scalable mobile advertising. In *Proceedings of the 5th ACM international workshop on Mobility in the evolving internet architecture*, pages 33–38. ACM, 2010.

[37] Michael Harkavy, J Doug Tygar, and Hiroaki Kikuchi. Electronic auctions with private bids. In *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*, volume 31. Boston, USA, 1998.

[38] Bernard J. Jansen, Danielle L. Booth, and Amanda Spink. Determining the informational, navigational, and transactional intent of web queries. *Information Processing & Management*, 44(3):1251–1266, May 2008.

[39] Anick Jesdanun. Ad Targeting Based on ISP Tracking Now in Doubt. *Associated Press*, September 2008.

[40] H. Kikuchi. (M+ 1) st-Price Auction Protocol. In *Proceedings of the 5th International Conference on Financial Cryptography*, pages 351–363. Springer-Verlag, 2002.

[41] H. Kikuchi, S. Hotta, K. Abe, and S. Nakanishi. Distributed auction servers resolving winner and winning bid without revealing privacy of bids. In *Parallel and Distributed Systems: Workshops, Seventh International Conference on, 2000*, pages 307–312. IEEE, 2000.

[42] Balachander Krishnamurthy and Craig Wills. On the Leakage of Personally Identifiable Information Via Online Social Networks.

[43] Balachander Krishnamurthy and Craig Wills. Privacy diffusion on the web: A longitudinal perspective. In *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*, Madrid, Spain, 2009.

[44] H. Lipmaa, N. Asokan, and V. Niemi. Secure Vickrey auctions without threshold trust. In *Proceedings of the 6th international conference on Financial cryptography*, pages 87–101. Springer-Verlag, 2002.

[45] Helger Lipmaa, N Asokan, and Valtteri Niemi. Secure vickrey auctions without threshold trust. In *Financial Cryptography*, pages 87–101. Springer, 2003.

[46] Microsoft, Inc. Start advertising on Yahoo! Search and Bing. `https://adcenter.microsoft.com/`.

[47] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[48] David C Parkes, Michael O Rabin, Stuart M Shieber, and Christopher Thorpe. Practical secrecy-preserving, verifiably correct and trustworthy auctions. *Electronic Commerce Research and Applications*, 7(3):294–312, 2008.

[49] Kurt Partridge, Manas A Pathak, Ersin Uzun, and Cong Wang. Picoda: Privacy-preserving smart coupon delivery architecture. In *PETS Symposium*, 2012.

[50] Adrian Perrig, Sean W. Smith, Dawn Xiaodong Song, and J. D. Tygar. Sam: A flexible and secure auction architecture using trusted hardware. In *Proceedings of the 15th International Parallel &Amp; Distributed Processing Symposium*, IPDPS '01, pages 170–, Washington, DC, USA, 2001. IEEE Computer Society.

[51] Michael O Rabin, Yishay Mansour, S Muthukrishnan, and Moti Yung. Strictly-black-box zero-knowledge and efficient validation of financial transactions. In *Automata, Languages, and Programming*, pages 738–749. Springer, 2012.

[52] Michael O Rabin, Rocco A Servedio, and Christopher Thorpe. Highly efficient secrecy-preserving proofs of correctness of computations and applications. In *LICS*, pages 63–76. Citeseer, 2007.

[53] Alexey Reznichenko and Paul Francis. Private-by-Design Advertising Meets the Real World. *Submitted to SIGCOMM '14*.

[54] Alexey Reznichenko, Saikat Guha, and Paul Francis. Auctions in Do-Not-Track Compliant Internet Advertising. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, Chicago, IL, October 2011.

[55] K. Sako. An Auction Protocol Which Hides Bids of Losers. In *Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography*, pages 422–432. Springer-Verlag, 2000.

[56] Stanford. Do Not Track Universal Web Tracking Opt-Out. `donottrack.us`.

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

[58] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. Adnostic: Privacy Preserving Targeted Advertising.

[59] Wikipedia. Usage share of operating systems. `http://en.wikipedia.org/wiki/Usage_share_of_operating_systems`.

Alexey Reznichenko

Max Planck Institute for Software Systems
Paul-Ehrlich-Strasse 26
D-67633 Kaiserslautern
Germany

Phone: +49 631 9303 9622
E-mail: areznich@mpi-sws.org
http://www.mpi-sws.org/~areznich

Research Interests

Networked and P2P systems, Privacy-by-Design

Education

Doctor of Engineering in Computer Science — Pursuing
Max Planck Institute for Software Systems, Kaiserslautern, Germany
Thesis: *Private-by-Design Advertising and Analytics: from Theory to Practice*
Advisor: Paul Francis

Master of Science in Computer Science — August 2009
Saarland University, Saarbrücken, Germany
Thesis: *Design and Implementation of a Collaborative Content Downloading System*
Advisor: Krishna Gummadi

Master of Science in Physics *cum laudé* — September 2007
Voronezh State University, Voronezh, Russia
Majors: Radio-Physics and Information Systems

Bachelor of Science in Physics *cum laudé* — June 2005
Voronezh State University, Voronezh, Russia
Majors: Radio-Physics and Information Systems

Academic Experience

Ph.D. Candidate advised by Paul Francis — June 2009 - April 2014
Max Planck Institute for Software Systems, Kaiserslautern, Germany

PrivAd: Practical Private-by-Design Advertising System
PrivAd is a private-by-design advertising system, which enables behavioral targeting without revealing user online behavior or user interest profiles to the ad network. Developed several online ad auction designs for PrivAd, which allow it to operate within the current business model (cost-per-click second price auction). Analyzed auction designs in terms of privacy risks, potential attack vectors and the impact on the targeting quality and revenue streams. Built a fully operational PrivAd prototype equipped with a differentially private analytics mechanism. Deployed and evaluated the system at scale with more than 13K opted-in users.

Graduate Student advised by Krishna Gummadi — September 2007 - May 2009
Saarland University and MPI-SWS, Saarbrücken, Germany

Collaborative BitTorrent (CBT)
Designed and implemented an extension to the BitTorrent protocol that allows socially connected users (i.e., friends who are willing to share their bandwidth with each other) to team up and engage in a collaborative distributed task to speed up the process of downloading a piece of content. Complementary resources provided by friends enabled CBT clients to achieve a significant improvement over vanilla protocol in the download performance. Built and evaluated a prototype on top of the Azureus BitTorrent client.

Industrial Experience

Software Engineering Intern October 2011 - March 2012
Google, Mountain View, USA
Contributed to design and implementation of Photon – a large-scale stateful distributed system for joining multiple continuously flowing streams of data with extremely high scalability and low latency. Photon is designed to withstand infrastructure degradation and datacenter-level outages without any manual intervention, providing significantly better fault-tolerance and lower maintenance overhead compared to typical single-datacenter systems. Photon is currently deployed within Google Advertising System to join multiple data streams generating joined logs critical for key business metrics.

Software Engineering Intern Summer 2011
Google, Zurich, Switzerland
Designed, implemented and deployed a service to provide advertisers with an additional feedback channel on effectiveness of their video ad campaigns on YoutTube. At the core of this service is a ranking algorithm that extracts useful signal from the bulk of user-generated comments.

Software Engineering Intern Summer 2010
Google, Munich, Germany
Contributed to the Scripting Layer for Android (SL4A) project whose goal is to enable users to create and execute applications written in scripting languages directly on their Android devices. Brought SL4A to a qualitatively new level by adding concurrent execution of multiple scripts and ability to distribute interpreters and scripts as standalone APKs, which enabled developers to distribute their application through Android Market. Additionally, enhanced UI and performance, added a number of supported APIs.

Software Developer February 2005 - March 2007
Siemens IT Solutions and Services, Voronezh, Russia
Developed Human Resources reporting tools in SAP R/3 with ABAP/4.

Publications

Private-by-Design Advertising Meets the Real World
A. Reznichenko and P. Francis
Submitted to ACM SIGCOMM 2014

Photon: Fault-tolerant and Scalable Joining of Continuous Data Streams
M. Singh, A. Gupta, H. Jiang, S. Das, T. Qiu, V. Basker, A. Reznichenko, S. Venkataraman, D. Ryabkov, A. Ananthanarayanan
in *Proceedings of the 2013 ACM SIGMOD/PODS Conference*, 2013

Towards Statistical Queries over Distributed Private User Data
R. Chen, A. Reznichenko, P. Francis and J. Gehrke
in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012

Auctions in Do-Not-Track Compliant Internet Advertising
A. Reznichenko, S. Guha and P. Francis
in *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, 2011

Serving Ads from localhost for Performance, Privacy, and Profit
S. Guha, A. Reznichenko, K. Tang, H. Haddadi and P. Francis
in *Proceedings of the 8th Workshop on Hot Topics in Networks (HotNets)*, 2009

Privad: Rearchitecting Online Advertising for Privacy
S. Guha, B. Cheng, A. Reznichenko, H. Haddadi, and P. Francis
Technical Report, 2009

Design and Implementation of a Collaborative Content Downloading System,
Master's Thesis, Department of Computer Science, Saarland University, July 2009

Teaching Experience	**Teaching Assistant**, Operating Systems, Saarland University	Summer 08
	Teaching Assistant, Software Engineering, Saarland University	Winter 07/08

Languages Russian (native), English (fluent), German (working knowledge)

References Available upon request