PPAD: Privacy Preserving Group-Based ADvertising in Online Social Networks

Sanaz Taheri Boshrooyeh
Koç University, İstanbul, Turkey
staheri14@ku.edu.tr

Alptekin Küpçü
Koç University, İstanbul, Turkey
akupcu@ku.edu.tr

Öznur Özkasap
Koç University, İstanbul, Turkey
oozkasap@ku.edu.tr

***Abstract*—Services provided as free by Online Social Networks (OSN) come with privacy concerns. Users' information kept by OSN providers are vulnerable to the risk of being sold to the advertising firms. To protect user privacy, existing proposals utilize data encryption, which prevents the providers from monetizing users' information. Therefore, the providers would not be financially motivated to establish secure OSN designs based on users' data encryption. Addressing these problems, we propose the first Privacy Preserving Group-Based Advertising (PPAD) system that gives monetizing ability for the OSN providers. PPAD performs profile and advertisement matching without requiring the users or advertisers to be online, and is shown to be secure in the presence of honest but curious servers that are allowed to create fake users or advertisers. We also present advertisement accuracy metrics under various system parameters providing a range of security-accuracy trade-offs.**

I. Introduction

Online Social Networks (OSN) such as Facebook, Twitter, and Google+ are in the center of people's attention due to the functionality and networking opportunities they offer. OSNs consist of three main entities: *Server, user*, and *advertiser*. *Server* supports the OSN's functions using its storage and computational resources. Users are able to share their personal information with each other and establish new friendships via OSN. Advertisers ask *Server*'s help to detect their target customers out of OSN's users. Despite the attractive services that OSNs offer to the users, sharing personal information with these networks raises privacy problems where servers monetize users' information by selling them to the advertising companies [23], [26]. To prohibit the accessibility of OSN providers to the plain information of users, secure OSN designs that employ data encryption are proposed [11], [10], [30], [32], [2], [3]. While these solutions provide tangible benefits to the users' privacy, they neglect the role of advertiser as part of the OSN, which results in monetizing *inability* for the server [31]. Thus, OSN servers are left with no financial motivation to establish such secure OSN services.

The lack of a convincing commercial model for secure OSNs is our main motivation to propose a Privacy Preserving ADvertising (PPAD) system for OSNs. PPAD can be incorporated into secure OSN designs (where the OSN's functionality meet data confidentiality) to provide advertising service. Yet, achieving the best of both worlds is impossible: we provide a trade-off between personalized advertising accuracy and user profile privacy. We first define these terms, explain why it is impossible to achieve some goals simultaneously, and show how we achieve a solution whose parameters can be tweaked for various settings.

In PPAD, we introduce the notion of **group-based advertising** on the encrypted data. By group-based advertising, we aim to cope with the security issues raised by the personalized counterparts [35], [20]. In fact, performing personalized advertising on the encrypted data will ultimately violate user privacy. The reason is that knowing that a particular advertising request (which is a set of attributes) is matched to an encrypted profile implies that the profile entails the attributes listed in that request. Therefore, although the matching is performed on the encrypted data, the server is able to learn the profile content i.e., user's attributes. This cannot be prevented using *any* (cryptographic) method unless one (unrealistically) assumes that the adversarial server cannot create fake advertisement requests targeting known attributes.

One remedy of this problem is that the final matching result must be computed in the encrypted format (server does not learn the result) and then the results are sent to the user to open and read. However, this approach is cumbersome as the user has to open (decrypt) all the matching results (which is linear in the total number of advertising requests) and retrieve the matched advertisements from the server in an oblivious way (which again incurs a high load).

Due to this privacy concern, we define a new advertising paradigm called group-based advertising. In short, we (randomly) partition users into groups of equal size at the registration phase. Then, each advertising request is compared to the profiles of a group of users and not a single user. The matching result indicates whether there exist some threshold-many target users among the group members. If it happens, then the advertising is shown to *all* the group members. Note that the matching result reveals neither the identity of the matched user nor the number of matched users but only the existence of at least threshold-many matches. By this method, the matching result is unlikable to an individual profile. We propose a formal security definition to capture this notion of unlinkability.

Another property of PPAD is to keep the advertising procedure **transparent** to the users/advertisers, similar to the insecure counterparts. That is, users and advertisers carry no overhead except uploading their data to the social network. Henceforth, the matching process is operated only by the server and needless to any constant online connection of the

user or the advertiser. Prior solutions [33], [14], [5], [16], [28], [21] fail to provide the transparency feature. User's involvement in the matching procedure adds an overhead to the user that grows linearly with the number of advertising requests. This overhead demotivates the user from participating in the PPAD protocol as the user is obliged to stay online until the server matches user's profile to the advertising requests. It also negatively affects the system's efficiency as the servers' working-time depends on the users' online time. In PPAD, users receive relevant advertisements, which are found by the server during the users' and advertisers' off-time. We enable this by utilizing a **privacy service provider** (PSP) that assists OSN providers to protect the privacy of their users. A PSP can be a non-profit or governmental entity that can help users and multiple providers achieve privacy-preserving advertising and can be implemented with low cost. Due to their fame and reputation, PSPs are assumed to be non-colluding parties and hence are used in similar privacy-concerned applications [34].

Our contributions in this paper are as follows.

- We propose PPAD advertising system that preserves user privacy and is applicable on the secure OSNs where users' information are encrypted.
- In contrast to the existing solutions where secure matching requires both user and advertiser to be permanently or simultaneously online, PPAD allows users and advertisers to be *offline* after the registration. Once the matching is performed offline by the server, the advertisement is shown to relevant users the next time they appear online (or via push notifications, etc.).
- We present a formal security definition for user privacy. We argue that a meaningful security definition in this setting must allow the adversarial server to control some advertisers and users. Our system is formally proven to be secure against the privacy service provider as well as the server that may additionally employ a number of fake users or advertisers.
- We define two performance metrics of *Target accuracy* and *Non-Target accuracy* to be used in group-based advertising systems. Using empirical analysis, we capture the effect of group size and threshold value on the system performance and discuss their security implications.

II. System Overview

Model: An overview of PPAD is shown in Figure 1. The participants are users, advertisers, and the OSN provider (*Server*) who gets help from a third party that is a privacy service provider (*PSP*). In PPAD, the advertiser specifies the attributes of its target users, and uploads an advertising request to the *Server* as a Bloom filter. We define an advertising request to be a conjunction of several attributes e.g., $\{Artist, Player, Scientist\}$. Users are (randomly) partitioned into groups of size k at the registration phase. As discussed, this grouping is *necessary* for user privacy, and must not be done based on similar interests. To preserve confidentiality, users **encrypt** their Bloom filters using additive homomorphic encryption and secret sharing techniques before submission to *Server*. To provide provable security, PPAD requires users to

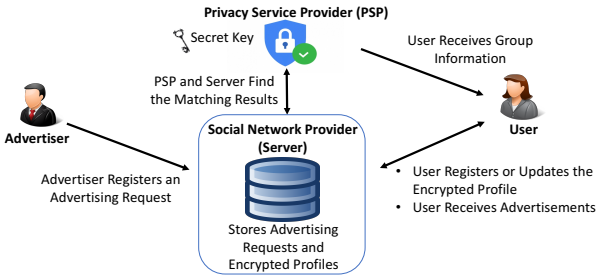

Fig. 1: PPAD system overview

encrypt their data using two different public keys PK_1 and PK_2. The decryption power i.e. the corresponding secret keys SK_1 and SK_2 are given only to *PSP*, but *PSP* never receives these individual profiles.

For each group, if at least threshold-many group members' profiles entail the same attributes listed in the advertising request, the group is marked as a target group. The advertisement is then presented to *all* members of the target groups (since advertising to a subset would require violating privacy). Afterwards, the social network provider charges the advertiser based on the number of target groups (hence users) found for its advertising requests. The group size and threshold are parameters that affect both advertisement accuracy and user privacy. We analyze this trade-off in Section V.

Security Goals: In PPAD, our security goal is to protect the link-ability of a successful match to a specific member. That is, when a group is marked as a target group, the server should not be able to say which members of that group exhibit the attributes in the advertising request.

However, it is important to realize that there is always an implicit and inherent privacy leakage in any advertising system, regardless of how secure it is designed. This leakage is that as soon as the server obtains the matching result, it understands the inclusion or exclusion of some specific attributes among the group members, although the matching is performed entirely on the encrypted profiles. In group-based advertising, the inclusion or exclusion of attributes in the group is unlinkable to a particular group member, while in the personalized counterpart, the matching result immediately breaks user privacy.

Security Assumptions: What we presume in PPAD is that the main adversary of user privacy is the social network provider, i.e., *Server*, (or an attacker controlling it) who may be curious to link the matching results of each group to the individual members. In this regard, the server may employ polynomially-many advertisers and take control of some of the users in each group. If the *Server* manages to control all but one member of some group, the same arguments against the impossibility of providing privacy in a personalized advertisement system apply. Therefore, we necessarily assume that at least two users per group are honest.

Moreover, the *Server* is assumed not to be able to collude with the privacy service provider, i.e., *PSP*. We believe that such a non-profit or governmental organization would not cooperate with the social network provider against user privacy due to the fame and reputation concerns. Hence, privacy

service providers are assumed to be non-colluding parties and are used in similar privacy-concerned applications [34]. In Section VI-B, we formally prove that neither *PSP* nor the *Server* would be able to violate user privacy.

Since we employ *PSP* as an external entity, it is important that it can be implemented with low cost, requiring minimal change in the OSN system. In PPAD, users contact *PSP* only during registration to receive some anonymous, non-personalized group parameters. Advertisers never need to contact *PSP*. Moreover, when *PSP* helps *Server* during the matching process, it performs two decryptions and some arithmetic operations per matching (6 milliseconds per matching). Section V-B presents more details on performance.

Preventing Compound Group Matching Although group matching preserves user privacy, the social network provider may learn the identity of the target users by arbitrarily combining profiles of users from different groups and analyzing the changes in the matching results. To avoid this misbehavior, users of each group are given zero-sum secret shares by *PSP*. They embed their secret shares in their encrypted profiles. Decryption of individual profiles with different embedded secret shares results in garbage values. Thus, any attempt by *Server* toward grouping arbitrarily chosen users' profiles fails. The only way to cancel out the secret sharing is to aggregate the profiles of users of the same group, as then the secret shares' summation would be zero. Thus, *Server* has to aggregate profiles of each group separately and sends the aggregated data to *PSP*. Finally, *PSP* decrypts and de-aggregates the data to find the number of matching users in the group. We enable aggregation and de-aggregation of profiles using super increasing sets (more details are presented in sections III and IV-A4).

Profile Update Insecurity Performing profile update in secure group-based advertising systems comes with a serious privacy issue, whose solution hugely degrades system efficiency. Essentially, when a member of group modifies her profile (by adding or removing some attributes), *Server* can analyze the changes in the matching results of that group (against advertising requests) before and after user's profile update and realize which attributes the user has modified in her profile. Also, the group-mates are vulnerable to the same security risk. Due to this security problem, if a user wants to modify her profile, she has to join a new group (similarly her group-mates), and the old group is now disfunctional. This is regardless of the underlying (cryptographic) tools employed.

Prior studies with the profile update functionality are not applicable to the context of advertising in social networks since they assume that the user does not share its profile with the server [33] or they employ an IP Proxy server [12] so that users anonymously add new preferences to their profiles. The former contradicts with the advertising transparency and degrades the performance. The latter is not applicable to OSNs since users access the social network via a particular account and hence the server observes that the update operation is done under a particular account.

III. Preliminaries

Bloom Filters: Bloom filters [7] are used to represent sets, and efficiently check whether an element belongs to a set. A Bloom filter is constructed with an array of p bits, initially zero, and d hash functions, $H_1(.)$,...,$H_d(.)$. p is called the size of the Bloom filter. To insert an element x_1 into the Bloom filter, all the hash functions are applied on x_1 (i.e. $i_1 = H_1(x_1)$,..., $i_d = H_d(x_1)$) and the array cells at indices corresponding to the hash outputs are set to 1. Testing the membership of an element is done by applying all the hash functions on it (similar to the insertion), and checking that whether all the corresponding indices are equal to 1. If one of them is not equal to 1, then that element does not belong to the set. Otherwise, with the false positive probability of $1-(1-((1-\frac{1}{p})^{d})^{e})^{d}$ the element belongs to the set, where e is the number of elements inserted into the Bloom filter. In the rest of the paper, $BFCreate(inSet)$ creates a Bloom filter with *inSet* being the set of input elements.

Super Increasing Set: A super increasing set [8] of length g is a series of g positive real numbers, $\{s_1, s_2, ..., s_g\}$, where each element is greater than the sum of its preceding elements i.e., $(\forall j \in \{2,...,g\} : s_j > \sum_{i=1}^{j-1} s_i)$.

Additive Homomorphic encryption scheme: A public key encryption scheme $(KeyGen, Enc, Dec)$ is called additive homomorphic [18] if for all m_0, m_1 from the message space $C_0 \odot C_1 = Enc(m_0) \odot Enc(m_1) = Enc(m_0 + m_1)$ where $\odot$ is an operation defined over ciphertexts. Example is Paillier encryption [27] where $\odot$ corresponds to the multiplication over ciphertexts.

Negligible Function: A function f is called negligible if $\forall$ positive polynomials $p(.)$ $\exists I$ *s.t.* $\forall i > I$ (where i is a real number): $f(i) < \frac{1}{p(i)}$.

Chosen plaintext attack (CPA) security: Consider the following experiment, $HEnc^{CPA}_{\pi,A}(\lambda)$, for the public-key encryption scheme $\pi = (KeyGen, Enc, Dec)$:

1) Run the $KeyGen(1^{\lambda})$ and obtain public key PK and private key SK. λ is the security parameter.
2) Adversary A, who is given PK, outputs two messages m_0 and m_1 of equal length from the message space.
3) Pick a random bit b and return to the adversary the encryption of m_b i.e. $Enc_{PK}(m_b)$.
4) A outputs a bit b'. If $b == b'$ the output of experiment is 1 indicating that A wins, otherwise A loses.

Definition 3.1: A public-key encryption scheme is CPA secure if $\forall$ probabilistic polynomial time adversary A, $\exists$ a negligible function $negl(\lambda)$ s.t. $Pr[HEnc^{CPA}_{\pi,A}(\lambda) = 1] \leq \frac{1}{2} + negl(\lambda)$

Secret Sharing Secret sharing [4] is a method to disseminate a secret among a set of parties. Consider zero as the secret, one way to create k shares of zero is to generate $k-1$ shares randomly ($SS_i, i \in \{1,...,k-1\}$) and then set the last share to $SS_k = 0 - \sum_{i=1}^{k-1} SS_i$ mod q where q indicates the modulus. The length of the secret shares must be long enough to hide the data content (longer than the maximum data size). We define $SSGen(k,q)$ as a function which generates k zero-sum secret shares out of the given message space (i.e., modulus) q.

IV. PPAD

A. Full Construction

This section presents our full construction, explaining which party runs which algorithm at which stage. Throughout the explanations, $x \leftarrow X$ demonstrates picking an element x uniformly at random from set X, and $||$ represents concatenation.

The **OSN initialization** is launched by *PSP* to generate system parameters. Users register their encrypted profiles in **User Registration**. A profile is a modified variant of a Bloom filter whose elements are separately encrypted under an additive homomorphic encryption scheme. Advertisers engage in **Advertiser Registration** protocol to submit an advertising request as a Bloom filter. The *Server* cooperates with *PSP* in the **Advertisement** protocol to find the target groups for each advertising request. In short, for every group and advertising request pair, the *Server* aggregates encrypted profiles of users in that group and sends the aggregate as well as the advertising request to *PSP*. Consequently, *PSP* checks if the group matches to the request and responds to *Server* accordingly.

1) OSN initialization (OSNInit)

Server: The *Server* initializes a database as *DB*.

PSP: *PSP* determines the security parameter 1^{λ} and the threshold value. It establishes an additive homomorphic encryption π=*(KeyGen,Enc,Dec)* scheme with message space *MSpace*, and generates (PK_1, SK_1) and (PK_2, SK_2) as two pairs of public and private keys. We need two sets of key pairs for the security proof to work. The reason will be apparent in Section VI-B.

2) User Registration (UReg)

UReg protocol is shown by Figure 2.

PSP: User connects to *PSP* via a secure and server authenticated channel to receive its group related information. Initially, *PSP* determines the group identifier *GID* of the user. *GID*s can be assigned according to the users' arrival i.e., the first k users are assigned to the first group and the second k users to the second group, etc. Note that a group needs to be full to be advertised to, since the secret shares will not sum up to zero otherwise. *PSP* generates a fresh set of k secret shares as $GSS = \{SS_1, ..., SS_k\}$ per group and assigns shares to the users.

Also, *PSP* assigns a delimiter, D, to each user such that D is unique among group-mates. Each user embeds its delimiter in the profile. The structure of delimiters is given in Equation 1. Delimiters exhibit the property of a superincreasing set and help in aggregation and de-aggregation of members' profiles during the advertisement protocol. *PSP* generates a set of k delimiters denoted by *DSet* once and uses them for every group.

$$\forall j \in \{1, ..., k\}, D_j > \sum_{i=1}^{j-1} D_i * p \tag{1}$$

User: To make a profile, users may enter their preferences into a well-structured form like a Facebook profile. However, the presentation of profile form to the users is an orthogonal issue to the PPAD. Ultimately, all the collected attributes (denoted by *AttSet*) are transformed to a modified version of a Bloom filter at the user side. In Algorithm 1, first a Bloom filter, Pf, is generated out of *AttSet*. Then, each bit i of Bloom

Algorithm 1 PCreate(AttSet, D, SS, PK)

1: $Pf = BFCreate(AttSet)$
2: $EPf = \{Enc_{PK}(Pf_i * D + SS)\}_{1 \le i \le p}$
3: **return** EPf

filter i.e., Pf_i is updated to $Pf_i * D + SS$. In words, we replace the 0-bit values of Bloom filter with user's secret share and the set bit values with the summation of the user's secret share and delimiter. This modification helps in two regards, first, to enable aggregation and de-aggregation by the help of delimiters, and second, to prevent compound group matching using secret shares (see the advertisement protocol). Finally, each modified element of Bloom filter is encrypted under the public encryption key *PK* given as input to the Algorithm 1.

The user selects its username *UName*, and generates two encrypted profiles EPf and $E\hat{P}f$ by running Algorithm 1 under two public keys PK_1 and PK_2, respectively. Then it uploads its encrypted profiles EPf, $E\hat{P}f$ alongside *UName* and *GID* to *Server*.

Server: *Server* receives the encrypted profiles and inserts them into the database *DB*.

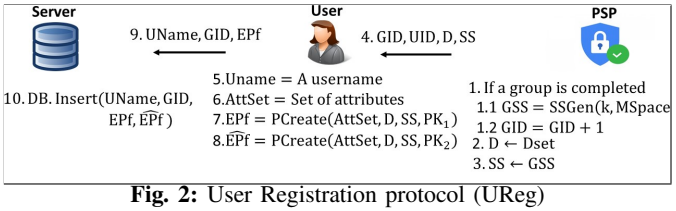

Fig. 2: User Registration protocol (UReg)

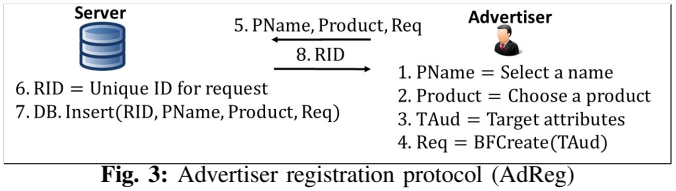

Fig. 3: Advertiser registration protocol (AdReg)

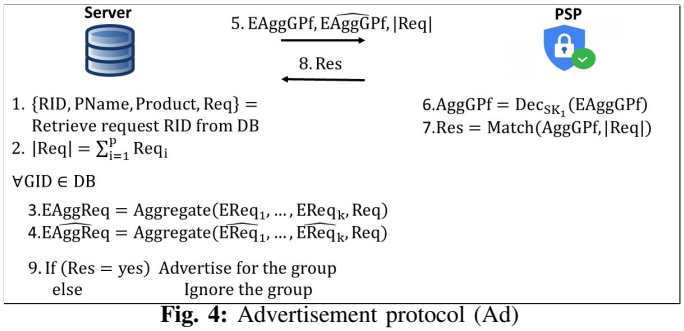

Fig. 4: Advertisement protocol (Ad)

3) Advertiser Registration (AdReg)

AdReg is shown in Figure 3. During *AdReg*, the advertiser registers its advertisement request under its name i.e. *PName* into the OSN. Advertiser specifies a set of attributes denoted by *TAud* for its target audience. The advertiser creates a request as a Bloom filter out of *TAud*. Then, it submits the Bloom filter, its name and the product to be advertised to *Server*. Advertising request preserves the AND operation between the targeted attributes. For example, a single request may target users with both attributes X **AND** Y. However, an advertising request which targets X **OR** Y must be split and submitted as two separate requests: one for X and the other

for Y. *Server* registers the request, assigns a unique request identifier *RID*, and sends *RID* to the advertiser.

4) Advertisement (Ad)

Figure 4 represents the *Ad* protocol. During the *Ad* protocol, *Server* first retrieves an advertising request i.e., $(RID, PName, Product, Req)$ from *DB*. Next, to find the matching between the request and each group of users, *Server* and *PSP* interact as follows (**both users and advertisers are offline during this procedure**, which is one of our main contributions):

***Server* Aggregate**: *Server* checks whether the advertising request is already matched against the group or not. If it is not matched, then *Server* retrieves profiles of group members and proceeds to the aggregation phase. As we already mentioned, the aggregation helps cancel out the secret shares embedded in users' profiles (as discussed in Section II, the purpose of secret shares is to prevent compound group matching).

To aggregate profiles, we utilize the fact that a profile Pf (which is a Bloom filter according to Algorithm 1) matches the advertising request *Req* if for every set bit position of *Req* the corresponding bit in Pf equals to 1. More formally, Pf is a target for *Req* if

$$\forall 1 \leq i \leq p \quad s.t. \quad Req_i = 1 \quad \rightarrow \quad Pf_i = 1 \tag{2}$$

Stated differently, Pf matches *Req* if the sum of set bit values of *Req* (we denote it by $|Req|$) equals to the sum of corresponding bit values in Pf. Due to this reason, we are only interested in the elements of a profile corresponding to the set bit positions of *Req*. As the first step of aggregation, we take out and sum up the encrypted elements of each profile in accordance with the set bit positions of *Req*. More formally,

$$A = \prod_{1 \leq i \leq p | Req_i = 1} EPf_i = Enc_{PK}(\sum_{1 \leq i \leq p | Req_i = 1} Pf_i * D + SS) \tag{3}$$

The second part of equality in Equation 3 holds due to utilization of an additively homomorphic encryption scheme. The *Server* performs this procedure for each profile of the group and obtains $A_1, ..., A_k$. Finally, the *Server* sums up A_j values and obtains

$$\begin{aligned} EAggGPf &= \prod_{j=1}^{k} A_j = Enc_{PK}(\sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} * D_j + SS_j) \\ &= Enc_{PK}(\sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} * D_j + \sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} SS_j) \\ &= Enc_{PK}(\sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} * D_j + \sum_{1 \leq i \leq p | Req_i = 1} \underbrace{\sum_{j=1}^{k} SS_j}_{=0}) \\ &= Enc_{PK}(\sum_{j=1}^{k} \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} * D_j) \\ &= Enc_{PK}(\sum_{j=1}^{k} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i}) \end{aligned} \tag{4}$$

As it can be easily verified from Equation 4, $EAggGPf$ is the encryption of sum of bit values of profiles (Bloom filters) multiplied by their corresponding delimiters (D_j). *PSP* will employ delimiters to extract individual matching results. The aggregation procedure is summarized in Algorithm 2.

Algorithm 2 Aggregate($EPf_1, ..., EPf_k, Req$)

1: **for** $1 \leq j \leq k$ **do**
2: $\quad A = \prod_{1 \leq i \leq p | Req_i = 1} EPf_{j,i}$
3: **end for**
4: $EAggGPf = \prod_{j=1}^{k} A_j$
5: **return** $EAggGPf$

PSP: *PSP* decrypts the aggregated data (*PSP* possesses two secret keys SK_1 and SK_2. It may use one or both of the keys to obtain the plaintext data. Since *Server* is assumed to be honest but curious, the encryption under PK_1 is consistent with the encryption under PK_2). Then, *PSP* counts the number of profiles matched to the request by proceeding as follows. We denote the decryption of $EAggGPf$ by $AggGPf$, that is

$$AggGPf = \sum_{j=1}^{k} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} \tag{5}$$

Let us reformulate $AggGPf$ by extracting the first term of the outer summation as

$$AggGPf = D_k * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{k,i} + \sum_{j=1}^{k-1} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} \tag{6}$$

Using Equation1, we know that

$$D_k > \sum_{j=1}^{k-1} D_j * p > \sum_{j=1}^{k-1} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i} \tag{7}$$

Therefore, if we divide $AggGPf$ by D_k we obtain the quotient and the remainder as indicated in Equation 8:

$$AggGPf = D_k * \underbrace{\sum_{1 \leq i \leq p | Req_i = 1} Pf_{k,i}}_{\text{Quotient}} + \underbrace{\sum_{j=1}^{k-1} D_j * \sum_{1 \leq i \leq p | Req_i = 1} Pf_{j,i}}_{\text{Remainder}} \tag{8}$$

The Quotient is the summation of bit values of Pf_k in accordance with the set bit positions of *Req*. Thus, if the Quotient equals to $|Req|$, then Pf_k is a match. *PSP* continues the iteratively on the Remainder, using D_{k-1} for the next division. At any step that the number of target users exceeds the threshold, *PSP* sends *Yes* to the *Server* and stops. If it was not the case, *PSP* responds *No*. The process of matching is presented in Algorithm 3. Note that, the delimiters enabled us to extract the matching result of individual members from the aggregate, but *PSP* does not have any information regarding the individual profiles and usernames.

Algorithm 3 $Match(AggGPf, |Req|)$

1: $count = 0$
2: **for** $D_j \in DSet,\ j \in \{k, ..., 1\}$ **do**
3: $\quad$ **if** $\frac{AggGPf}{D_j} == |Req|$ **then**
4: $\quad\quad count = count + 1$
5: $\quad\quad$ **if** $count == Threshold$ **then return** *Yes*
6: $\quad\quad$ **end if**
7: $\quad$ **end if**
8: $\quad AggGPf = AggGPf \mod D_j$
9: **end for**
10: **return** *No*

***Server* Show**: Based on the *PSP*'s response, *Server* either advertises the *Product* for all the members of the group, or

Overhead\Entity	User	Advertiser	*Server*	*PSP*
User registration	$O(p)$	-	-	-
Advertiser registration	-	O(1)	-	-
Advertisement	-	-	$O(k.\|Req\|)$	$O(1)$

(a) Running Time (per matching)

Overhead\Entity	User	Advertiser	*Server*	*PSP*
User registration	$O(p)$	-	$O(p)$	-
Advertiser registration	-	$O(p)$	$O(p)$	
Advertisement	-	-	$O(1)$	$O(1)$

(b) Communication Complexity (per matching)

TABLE I (a) Running time based on the homomorphic operations. (b) Communication complexities (number of message transmissions). k: number of users per group. p: size of Bloom filter. $|Req|$: number of set bits in each advertising request, which is $O(e.d)$ where e is the number of attributes in each request and d is the number of hash functions used in the Bloom filter construction.

skips that group. The total number of target groups is counted and stored for monetizing purposes.

For each advertisement request, the protocol above is repeated for each group that is not yet matched with the request. Since we prevent group compounding, matchings can be performed in *parallel*. This means, while the computational complexity scales, communication rounds do *not* need to increase with the number of yet unmatched groups.

V. Performance

A. Asymptotic Performance

Table Ia shows the computational overhead of each entity in PPAD based on the number of homomorphic operations. n corresponds to the total number of users in the OSN, and m corresponds to the total number of advertising requests.

Users The user carries $O(p)$ computational overhead, only once, to element-wise encrypt its Bloom filter under *PSP*'s public keys where p is Bloom filter's size.

Advertisers The advertiser does not perform any cryptographic operation.

Server The running time complexity of *Server* to aggregate users' profiles within their corresponding groups is $O(k.|Req|)$ where $|Req|$ is the number of set bits in the Bloom filter of the advertising request. *Server* carries this overhead per group and advertising request pair. In total, there are $\frac{n}{k}$ groups and m advertising requests, hence the total overhead of *Server* yields to $O(m.\frac{n}{k}.k.|Req|) = O(m.n.|Req|)$.

***PSP*:** For a single matching, *PSP* has the running time complexity of $O(1)$ (to decrypt the group aggregated profiles). *PSP* performs the matching procedure per group and advertising request pair, which in total leads to the complexity of $O(m.\frac{n}{k})$ for its lifetime.

Table Ib demonstrates the communication complexity of each entity during the execution of each protocol. Users and advertisers need to share their Bloom filters with *Server*. Thus, $O(p)$ message transmission is required. *Server* and *PSP* communicate $O(1)$ messages to check the matching between a single group and an advertising request. For n users ($\frac{n}{k}$ groups) the total communication overhead of *Server* and *PSP* is $O(\frac{n}{k})$.

B. Concrete Performance

The running times are computed by executing PPAD over 1000 randomly generated profiles of 400 attributes (based on our personal experience of Facebook advertising, 400 attributes is approximately the maximum number) under the group size of 5. The advertising request is presumed to have 30 attributes (for randomly generated profiles, almost no match is found for an advertisement with more than 30 attributes). The results are taken on an Intel i5 2.60 GHz CPU, using 2048 bit keys for Paillier encryption scheme. Under this configuration, *Server* matches a single advertising request to a single group in 50 ms whereas running time of *PSP* is 6 ms, which is almost an order of magnitude better than that of *Server*. Profile creation time is 750 ms (done once per user) and creating an advertising request takes 0.5 ms.

C. Advertisement Accuracy Metrics

In order to analyze the effect of different group sizes and threshold values on the advertising performance, we define two performance metrics, namely ***Target accuracy*** and ***Non-Target accuracy***.

Target accuracy indicates the fraction of target users who are served by the advertisement, as formulated in Equation 9

$$\textbf{\textit{Target accuracy}} = \frac{\text{Number of target users served by the advertisement}}{\text{Total number of target users}} \tag{9}$$

This metric is in compliance with the advertiser desire who wants to reach as many target users as possible. Due to the nature of group-based advertising, the *Target accuracy* is not always 100% since the target users in groups with fewer than threshold-many target users are not shown the advertisement.

Non-Target accuracy as shown in Equation 10 is the fraction of non-target users that are **not** served an (irrelevant) advertisement.

$$\textbf{\textit{Non-Target accuracy}} = \frac{\text{Number of non-target users not served by the advertisement}}{\text{Total number of non-target users}} \tag{10}$$

The higher value of this metric indicates that users are less likely to be shown irrelevant advertisement (hence more accurate is the advertising and less disturbing).

Note that the *Target accuracy* and *Non-Target accuracy* are meaningful only in the group-based advertising paradigm and not in personalized counterparts (where both measures are perfectly satisfied with the cost of privacy loss).

We additionally define the notion of **target coverage**, which is the fraction of target users, as follows:

$$\textbf{Target Coverage} = \frac{\text{Number of target users}}{\text{Total number of users}} \tag{11}$$

The coverage value depends on the attribute distribution in profiles as well as the content of the advertisement. In our experiments, we target various levels of coverage and analyze the effect of our system parameters.

D. Advertisement Accuracy Results

We explore the effect of group size and threshold value on the *Target accuracy* and *Non-Target accuracy*. The results are taken over 100,000 profiles with three different target coverage values (10%, 50% and 90%) as demonstrated in Figure 5. The results present that under a specific group size, increasing the threshold value improves the *Non-Target accuracy*. This behavior is expected since having a higher

threshold guarantees that more target customers are in the target groups (compared to the lower thresholds). Hence in such settings, the higher percentage of target group members are real target customers i.e., the *Non-Target accuracy* is higher. On the contrary, the *Target accuracy* has the inverse relation with the threshold value. Indeed, higher threshold imposes more constraint on the group for being selected as a target. Consequently, the advertiser loses some of his target customers in the groups which do not have enough target users.

On the other hand, with a fixed threshold, as the group size increases, *Target accuracy* increases but *Non-Target accuracy* decreases. This happens for all target coverages, since in a larger group with the same threshold, it is easier to find matching groups, but it also means that potentially more non-target users are shown an irrelevant advertisement.

By inspecting the behavior of *Target accuracy* and *Non-Target accuracy*, we find out that a perfect balance between these two metrics is met when the ratio of the threshold to the group size i.e., $\frac{Thr}{\text{Group Size}}$ is close to the target coverage. We refer to this threshold value as **"balanced threshold"**. For instance, under the target coverage 50% and group size 19, the balanced threshold is 10 with $\frac{10}{19} = 0.52 \approx 0.5$. At this balance threshold, *Target accuracy* and *Non-Target accuracy* are 58% and 60%, respectively. We refer to the accuracy achieved at the balance threshold by **balanced accuracy**. In Figure 5, the x coordinate of the point where two curves of the same color (i.e., same group size) collide indicates the balanced threshold and the accuracy at that point (y coordinate) is the balanced accuracy. After the balanced threshold, the *Target accuracy* drops while the *Non-Target accuracy* increases. The inverse occurs for values less than the balanced threshold.

The simulation results demonstrate that as the group size increases, the balanced accuracy degrades. For example, under the target coverage of 50%, the balanced accuracy of group size 7 (at balanced threshold 4) is 65% whereas in group size 19 (at balanced threshold 10) it drops to 58%. The correctness of this fact can be verified by coverage 10 and 90 as well. This implies that smaller group sizes are better for accuracy at their respective balanced thresholds.

In general, threshold being equal to group size k would mean that all users in a matched group have the same attributes in the advertisement in common. Similarly, threshold of 1 where the advertisement is not matched would reveal that no user in that group contains all the attributes in the advertisement. Such leakages are independent of the underlying methodology, and hence are not analyzed, but should be considered when selecting the parameters.

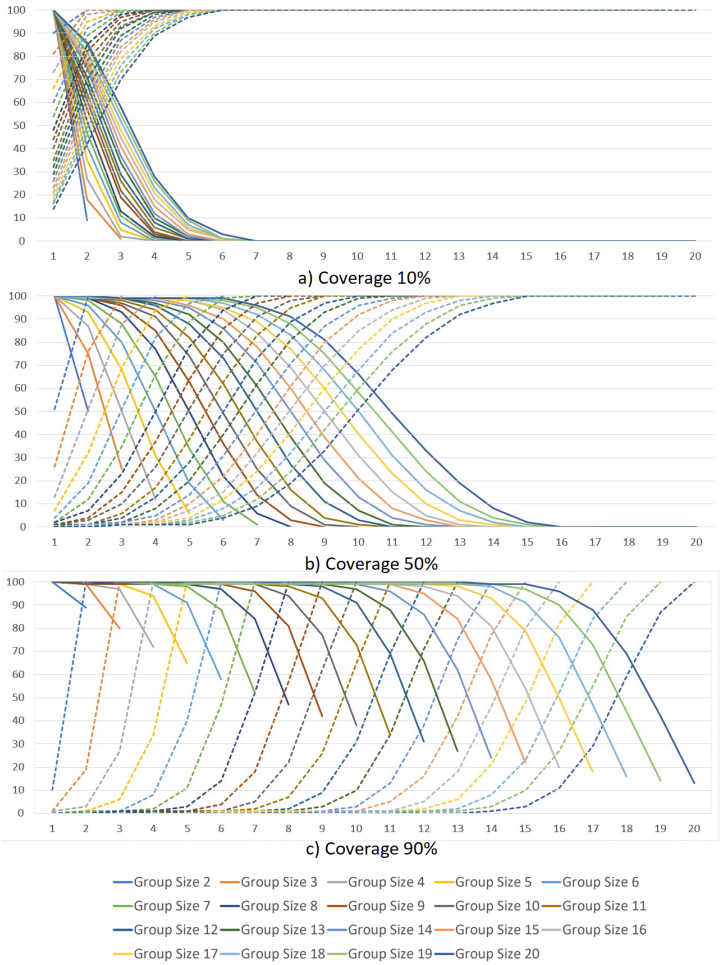

Fig. 5: *Target accuracy* and *Non-Target accuracy* vs threshold for group sizes 2-20. Dashed curves represent *Non-Target accuracy* and solid ones the *Target accuracy*. X axis: threshold. Y axis: accuracy

VI. Security

A. Security Definition

PPAD preserves user privacy if no adversary can link a successful group-matching result to a particular group member. In another word, the advertising result should not help an adversary to identify which user possesses (or does not possess) which attributes. The adversary controls *either Server* or *PSP* (since they are non-colluding), together with some users and advertisers. The adversary is challenged to break the user's privacy in a single group. This challenge is modeled as a game played between a challenger and the adversary A. Since the groups are independent of each other and the protocol is the same for every group, the failure of the adversary in this game implies that PPAD preserves privacy of all the users.

In this game, the adversary is allowed to control $k-t$ users where k is the group size and t is the number of honest users in that group. Adversary registers $k-t$ users of the group into the system and receives all of their secret information. Assume $UName_1$, ..., $UName_t$ are the usernames of the honest users. Adversary is asked to select t sets of attributes, $AttSet_1$,...,$AttSet_t$. The challenger randomly and privately assigns the attribute sets to the usernames and registers them into the OSN. Then, the adversary is allowed to register polynomially-many advertising requests and obtain the results of matching between the requests and the group. Finally, the adversary is challenged to guess which attribute set is assigned to which username. To win the game and break security, the adversary needs to perform noticeably better than the random guessing probability of $\frac{1}{t}$.

Observe that as t gets smaller, the adversary has more

control over the group, and hence has more power. But, for $t = 1$, the adversary wins the game with the probability of 1; therefore $t = 2$ is the minimum feasible value.

$UPrivacy_A(\lambda)$: In this game, the challenger acts as the honest users and honest advertisers. One of the *Server* or *PSP* is run by the challenger while the other one is controlled by the adversary *A*. Adversary *A* is honest but curious, and may control polynomially-many advertisers and k-2 users per group. The game is played within one group.

1) *A* runs OSNInit with the challenger.
2) Query phase 1:
   a) *A* runs *UReg* protocol, acting as a user, with the challenger.
   b) *A* specifies user's inputs *UName*,*AttSet* and asks the challenger to run *UReg* protocol over the given inputs. Challenger acts as user.

   Part (a) allows the adversary register users fully under her control. Part (b) allows the adversary to register honest users whose usernames and profiles are known to the adversary.
3) Challenge phase:
   a) *A* sends two usernames i.e. $Uname_0$ and $UName_1$ and two different sets of attributes, $AttSet_0$ and $AttSet_1$. $UName_0$ and $UName_1$ are never registered in any query phase.
   b) Challenger picks a bit randomly, $b \leftarrow \{0,1\}$, and executes the *UReg* protocol for ($Uname_0$, $AttSet_b$) and ($UName_1$, $AttSet_{\hat{b}}$) on behalf of users. $\hat{b}$ is the complement of b.
4) *A* repeats the query phase 1 until all the k users of the group are registered into the OSN.
5) Query phase 2:
   a) *A* creates and registers an advertising request by executing *AdReg* protocol acting as the advertiser.
   b) *A* selects the inputs of the advertiser for *AdReg* protocol and asks the challenger to execute *AdReg* protocol as an advertiser.

   Similar to query phase 1, part (a) allows the adversary register advertisement requests fully under her control. Part (b) allows the adversary to register honest advertising requests of which are known to the adversary.
6) Query phase 3: *A* executes the *Ad* protocol with the challenger for an advertising request registered as *RID*.
7) *A* may adaptively repeat the query phase 2 and 3 polynomially many times.
8) *A* guesses a bit $b^{'}$. If $b = b^{'}$ the output of game is 1 (*A* wins), otherwise 0 (*A* loses).

Definition 6.1: An *OSN* with the (*OSNInit*, *UReg*, *AdReg*, *Ad*) protocols preserves the user's privacy, if for every probabilistic polynomial time (PPT) adversary A, there exists a negligible function $negl(\lambda)$, where λ is the security parameter, such that: $Pr[UPrivacy_A(\lambda) = 1] \leq \frac{1}{2} + negl(\lambda)$

B. User Privacy Against Server

The security of our design against *Server* relies on the CPA-security of the underlying encryption scheme. We show that if a PPT adversary *A* can win the *UPrivacy* game with non-negligible advantage, then we can construct a PPT adversary *B* who runs *A* as a subroutine and breaks the CPA-security of the encryption scheme. At a high level, since group matching does not reveal the identity of the targeted users, but only provides a *Yes/No* type answer, *Server* cannot map users' profiles and usernames using the result of advertising. Therefore, the information of *Server* is restricted to the encrypted data. Thus, the success of adversary *A* in *UPrivacy* game implies that *B* can distinguish between the encrypted profiles of users. This means that encryption scheme is not CPA-secure which is a contradiction to the initial assumption. So, using a CPA-secure encryption scheme, our design is secure against *Server*. The formal proof follows.

1) Formal Proof

In PPAD, two key pairs for *PSP* are considered i.e., (PK_1, SK_1) and (PK_2, SK_2). Each data is encrypted under both keys. Adversary *B* aims at breaking the CPA-security of the encryption scheme associated with the first key pair i.e., (PK_1, SK_1).

Theorem 6.1: If the homomorphic encryption scheme π is CPA secure, then PPAD preserves user privacy.

Proof. If there exists a PPT adversary *A* who can win the $UPrivacy_A(\lambda)$ with probability $\frac{1}{2} + \varepsilon(\lambda)$ where $\varepsilon(\lambda)$ is non-negligible, then we can construct a PPT adversary *B* who can win the $HEn^{CPA}_{\pi,B}(\lambda)$ with $\frac{1}{2} + \varepsilon(\lambda)$ probability. Assume a PPT adversary *A* and define: $Pr[UPrivacy_A(\lambda) = 1] = \frac{1}{2} + \varepsilon(\lambda)$. Now consider the PPT adversary *B* who attempts to break $HEnc^{CPA}_{\pi,B}(\lambda)$.

Adversary *B*:

B is given the security parameter λ and the public key PK_1 by the CPA challenger.

1) *B* executes *OSNInit* protocol with *A*. *B* creates PK_2, SK_2, delimiter set *DSet* and a set of *k* secret shares i.e. $GSS = SSGen(k, MSpace)$.
2) Query phase 1 is executed.
3) Challenge phase: *A* sends two usernames $UName_1$ and $UName_2$ and two sets of attributes $AttSet_1$ and $AttSet_2$. *B* picks randomly two unique delimiters from *DSet* denoted by D_1 and D_2 and two unique secret shares SS_1 and SS_2. *B* creates two bloom filters out of Att_1 and Att_2 denoted by Pf_1 and Pf_2, respectively. *B* replaces each bit value b_i in Pf_1 (and Pf_2), by $b_i * D_1 + SS_1$ (and $b_i * D_2 + SS_2$). The modified bloom filters are denoted by $\hat{Pf_1}$ and $\hat{Pf_2}$, respectively. *B* sends $m_0 = \hat{Pf_0}||\hat{Pf_1}$ and $m_1 = \hat{Pf_1}||\hat{Pf_0}$ to the CPA challenger and receives a vector of ciphertexts shown by $C = \{C_1, ..., C_p, C_{p+1}, ..., C_{2p}\}$. Set $EPf_1 = \{C_1, ..., C_p\}$ and $EPf_2 = \{C_{p+1}, ..., C_{2p}\}$. *B* creates two other profiles encrypted under PK_2 denoted by $E\hat{P}f_1 = PCreate(AttSet_1, D_1, SS_1, PK_2)$ and $E\hat{P}f_2 = PCreate(AttSet_2, D_2, SS_2, PK_2)$. *B* registers $EPf_1, E\hat{P}f_1$ and $EPf_2, E\hat{P}f_2$ for $UName_1$ and $UName_2$, respectively. Although, EPf_1 and $E\hat{P}f_1$ (likewise EPf_2 and $E\hat{P}f_2$) may not contain the same attributes, this would not be revealed to *A* due to the CPA-security of encryption scheme associated with PK_2.

4) A repeats query phase 1 to register all the k users of the group.
5) Query phase 2 is executed. The set of all the registered requests is denoted by RQ.
6) Query phase 3: A runs the Ad protocol for a request Req out of RQ. A as $Server$ outputs the aggregation of group's profiles i.e., $EAggGPf$,$EAg\hat{g}GPf$, and the number of set bits in the advertising request i.e., $|Req|$. $EAggGPf$ is encrypted under PK_1 whereas $EAg\hat{g}GPf$ is encrypted under PK_2.
   Here is the reason why PPAD utilizes two key pairs. Both encryptions under PK_1 and PK_2 contain the same data since the adversary A is honest but curious (i.e. it follows the protocol description honestly). Moreover, the order of attributes' assignments to the users (either $AttSet_1$ and $AttSet_2$ are assigned to $UName_1$ and $UName_2$ respectively, or vice versa) does not affect the aggregated result and consequently the group matching result. Despite the lack of SK_1 at the side of adversary B, having SK_2 enables A to perform the advertising procedure correctly. Thus, B obtains the plain format of aggregated group's profiles by decrypting via SK_2. B performs Algorithm 3 and obtains the matching result. It sends the Yes or No response to A.
7) Query phases 2 and 3 may be adaptively repeated polynomially many times.
8) A outputs a bit $b^{'}$. B outputs whatever A outputs.

Since B outputs whatever A outputs, B has the same advantage of A to break the CPA security of the encryption scheme. If A's advantage $\varepsilon(\lambda)$ is non-negligible, then B also breaks the CPA-security of the encryption scheme with this non-negligible advantage.

C. User Privacy Against PSP

PPAD provides information-theoretic privacy for users against PSP. PSP never receives the usernames during the execution of any protocol. This implies the inability of PSP to obtain a mapping between the data contents, i.e., attributes, and the identity of the data owners.

VII. Related Works

A. Secure Online Behavioral Advertising (SOBA)

In SOBA models, a broker is connected to a set of publishers who are web page owners. The broker creates a behavioral profile per user according to the user's visits on those pages. Broker monetizes by putting the advertiser's products on the publishers' web pages according to the users' behavioral profiles. We classify SOBA models as publisher-subscriber and push-based designs as shown in Table II.a (also considering PPAD applied to such a setting). In publisher-subscriber designs [33], [14], [12], users subscribe to the advertisers' products. In push-based designs [5], [1] a server receives both the users' profiles and advertising requests, and advertises each product for a set of target users. Some SOBA studies require users or advertisers to be online during the advertising procedure [33], [14], [5], while others allow them to remain offline [1]. Some studies [14], [5] enforce direct communication between users and advertisers. Outsourced profiling [5] does not consider user privacy. ObliviAd [1] relies on a trusted hardware (CPU) to protect user privacy.

B. Server Assisted Private Set Intersection (PSI)

In the PSI problem, two parties who have two different sets of elements execute a protocol to find the intersection of their sets. In the server assisted variant of PSI, a server helps the parties to find the intersection of the sets, improving efficiency. Table II.b summarizes the comparison between PPAD and papers of the server-assisted PSI concept. In the server-assisted PSI studies, the role of the server is to reduce the workload of parties by carrying the main portion of the computations. However, at least one party still needs to be involved per protocol execution as in [16], [28], [21] and the oblivious service provider method of [20]. Parties also need to have direct communication for sharing some secret information before the execution of the intersection (advertisement) protocol. The public output method of [20] and [35] support offline users and advertisers, but they fail to protect the privacy of users. In fact, their solution is vulnerable to the plaintext guess attack where the server guesses some elements and checks whether they belong to the user's and advertiser's sets or not.

C. Server Assisted Two-Party Computation (2PC)

In server assisted 2PC protocols, two parties, with the help of a third party, compute a function over their respective inputs while no party learns the other party's input. Server-assisted 2PC solutions either employ a server to guarantee the fairness of the protocol execution [13], [22], [24] or to ease the other parties' duties by delivering the main computation overhead to the server. However, users and advertisers are required to be online and provide some information per function evaluation (advertisement in this case) [17], [19], [13], [25], [15], [9], [6]. [29] proposed a solution which mitigates the necessity of online users and advertisers by applying two servers, similar to our approach. But, the number of messages transferred between two servers depends on the function definition (number of multiplication operations), whereas PPAD supports constant communication compexity between two servers.

VIII. Conclusion and Future Work

In this paper, we proposed the first privacy preserving advertising system PPAD for secure OSNs with transparency and group advertising. PPAD protects users' privacy by employing an external non-colluding privacy service provider. We proposed a security definition and formally proved the security of our design under the honest-but-curious adversarial model where the adversary is additionally allowed to control some (fake) advertisers and users. As future work, our aim is to extend PPAD to be secure against fully malicious adversaries, and to efficiently support any Boolean function of the attributes in a single advertising request. We also plan to improve our solution to reduce the computational cost associated with profile updates. We believe PPAD constitutes an important first step regarding monetization for secure OSNs.

Acknowledgements

We acknowledge the support of the Turkish Academy of Sciences, Royal Society of UK Newton Advanced Fellowship NA140464, and EU COST Action IC1306.

Type	Method	Offline User	Offline Advertiser	EDC	User Privacy	No IP Proxy	No Trusted-Hardware	Sec-Def
PS	Adnostic [33]	✗	✓	✓	✓	✓	✓	✗
	Targeted advertising [14]	✗	✗	✗	✓	✓	✓	✓
	Privad [12]	✓	✓	✓	✓	✗	✓	✗
PB	Outsourced profiling [5]	✗	✗	✗	✗	✓	✓	✗
	ObliviAd [1]	✓	✓	✓	✓	✓	✗	✓
	PPAD	✓	✓	✓	✓	✓	✓	✓

(a)

Method	Offline User	Offline Advertiser	EDC	PGA
Privacy aware Genome Mining [28], Scaling PSI to billion elements [16]	✗	✗	✗	✓
VDSI [35],Collision Resistant outsourcing PSI [20] public output	✓	✓	✓	✗
Collision Resistant outsourcing PSI [20] Oblivious Service Provider	✗	✗	✓	✓
Outsourced PSI using homomorphic encryption [21]	✓	✗	✓	✓
PPAD	✓	✓	✓	✓

(b)

TABLE II (a) SOBAs vs. PPAD. (b) Server Assisted PSIs vs. PPAD. PS: publisher-subscriber, PB: push-based. EDC: Elimination of direct communication between users and advertisers. Sec-Def: Existence of formal security definition and proof. PGA: Security against plaintext guess attack.

REFERENCES

[1] M. Backes, A. Kate, M. Maffei, and K. Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *Security and Privacy (SP)*. IEEE, 2012.

[2] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an online social network with user-defined privacy. In *ACM SIGCOMM*, 2009.

[3] A. Barenghi, M. Beretta, A. Di Federico, and G. Pelosi. Snake: An end-to-end encrypted online social network. In *ICESS*. IEEE, 2014.

[4] A. Beimel. Secret-sharing schemes: a survey. Springer, 2011.

[5] D. Biswas, S. Haller, and F. Kerschbaum. Privacy-preserving outsourced profiling. In *CEC*. IEEE, 2010.

[6] M. Blanton and F. Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. *PET*, 2016.

[7] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. 1970.

[8] A. A. Bruen, M. A. Forcinito, A. G. Konheim, C. Cobb, A. Young, M. Yung, and D. Hook. Applied cryptography: protocols, algorithms, and source code in c. 1996.

[9] H. Carter, B. Mood, P. Traynor, and K. Butler. Outsourcing secure two-party computation as a black box. In *Cryptology and Network Security*. 2015.

[10] E. De Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the time of twitter. In *Security and Privacy (SP)*. IEEE, 2012.

[11] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten. Social networking with frientegrity: Privacy and integrity with an untrusted provider. In *USENIX*, 2012.

[12] S. Guha, B. Cheng, and P. Francis. Privad: practical privacy in online advertising. In *NSDI*, 2011.

[13] A. Herzberg and H. Shulman. Oblivious and fair server-aided two-party computation. *Information Security Technical Report*, 2013.

[14] A. Juels. Targeted advertising... and privacy too. In *CT-RSA*. 2001.

[15] S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011.

[16] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. Scaling private set intersection to billion-element sets. In *FC*. 2014.

[17] S. Kamara, P. Mohassel, and B. Riva. Salus: a system for server-aided secure function evaluation. In *CCS*. ACM, 2012.

[18] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC press, 2014.

[19] F. Kerschbaum. Adapting privacy-preserving computation to the service provider model. In *CSE*. IEEE, 2009.

[20] F. Kerschbaum. Collusion-resistant outsourcing of private set intersection. In *Applied Computing*. ACM, 2012.

[21] F. Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *CCS*. ACM, 2012.

[22] H. Kılınç and A. Küpçü. Efficiently making secure two-party computation fair. In *FC*, 2016.

[23] B. Krishnamurthy and C. E. Wills. Characterizing privacy in online social networks. In *WOSN*. ACM, 2008.

[24] A. Küpçü and P. Mohassel. Fast optimistically fair cut-and-choose 2pc. In *FC*, 2016.

[25] P. Mohassel, O. Orobets, and B. Riva. Efficient server-aided 2pc for mobile phones. *PET*, 2015.

[26] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *Security and Privacy*. IEEE, 2009.

[27] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[28] C. Patsakis, A. Zigomitros, and A. Solanas. Privacy-aware genome mining: Server-assisted protocols for private set intersection and pattern matching. In *CBMS*. IEEE, 2015.

[29] A. Peter, E. Tews, and S. Katzenbeisser. Efficiently outsourcing multiparty computation under multiple keys. *IEEE T-IFS*, 2013.

[30] J. Sun, X. Zhu, and Y. Fang. A privacy-preserving scheme for online social networks with efficient revocation. In *INFOCOM*. IEEE, 2010.

[31] S. Taheri-Boshrooyeh, A. Küpçü, and Ö. Özkasap. Security and privacy of distributed online social networks. In *IEEE ICDCSW*, 2015.

[32] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: better privacy for social networks. In *CoNEXT*. ACM, 2009.

[33] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *NDSS*, 2010.

[34] T. Veugen, R. de Haan, R. Cramer, and F. Muller. A framework for secure computations with two non-colluding servers and multiple clients, applied to recommendations. *IEEE T-IFS*, 2015.

[35] Q. Zheng and S. Xu. Verifiable delegated set intersection operations on outsourced encrypted data. In *IC2E*. IEEE, 2015.