Privacy-preserving User Matching

Paolo Gasti
Department of Computer Science
New York Institute of Technology
New York, USA
pgasti@nyit.edu

Kasper B. Rasmussen
Department of Computer Science
University of Oxford
Oxford, UK
kasper.rasmussen@cs.ox.ac.uk

ABSTRACT

Matching two or more users with related interests is an important and general primitive, applicable to a wide range of scenarios including job hunting, friend finding, and dating services. Existing on-line matching services requires participants to trust a third party server with their preferences. This raises security and privacy issues. In this paper, we tackle this problem by introducing two privacy-preserving protocols: *server-led matching* and *user-led matching*. In the first protocol, potential matching pairs (e.g., users, companies) are selected by the server, which collects and combines each party's preference. In the second, entities are allowed to express their preference for any party—regardless of whether the other party is known to the server. With server-led matching, users reveal no information to the server; the server's role is simply to relay messages. In user-led matching, the server only learns which users match. Our protocols are scalable, i.e., preferences can be matched in constant time. We formally define security and functionality requirements for generic server-led and user-led matching protocols, and provide security proofs for our instantiations within this framework.

1. INTRODUCTION

User matching is the process by which two parties, say Alice and Bob, express their mutual interest by issuing commitments to each other. Matching parties with related interests is an important and general primitive, applicable to a wide range of scenarios including job hunting, friend finding, dating services, and on-line bidding. In broad terms, there exist two types of user matching: one where the server is in charge of selecting two parties (Alice and Bob), who then indicate whether there is mutual interest (i.e., they issue yes or no commitments); and another where users independently select a party they are interested in, and issue a commitment for that party. We refer to the two types as *server-led matching* and *user-led matching* respectively.

In a non-cryptographic setting, these functionalities are usually achieved by relying on a trusted server that acts as an intermediary between the Alice and Bob (e.g., [8, 12, 16, 20, 25]). The server has full knowledge of the preferences expressed by all users, which causes considerable privacy issues: for server-led matching, the server can leak (either intentionally, or accidentally) users' interests—or lack thereof; for user-led matching, the server can leak un-matched commitments, with possibly professional and social repercussions for the users. The goal of privacy-preserving user matching is to implement server-led and user-led matching without revealing user preferences to the server, and possibly to the other user, unless the two users independently select each other. In this setting, the trusted server is replaced with a secure cryptographic protocol, run by Alice, Bob, and an untrusted server.

WPES'15, October 12, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3820-2/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2808138.2808148

Our **server-led protocol** allows users to upload a public profile (e.g., a nickname and a picture, a set of skills, etc.) which the server then uses to find *candidate pairs*, for matching. When a pair (Alice, Bob) has been identified, the server sends Bob's profile to Alice and Alice's profile to Bob. The parties decide whether they want to commit to each other, and thus learn the outcome of the protocol, which is either (i) both users decided to commit; or (ii) at least one of the users decided not to commit. No additional information is revealed to the parties (e.g., if Alice does not commit to Bob, she does not learn if Bob committed to her). The matching server never learns the choice of either participant (i.e., whether their commitment was yes or no), so no information about the user's preferences is revealed.

In our **user-led protocol**, users discover each other outside of the matching system, e.g., by knowing each other in real life or through social media. Then, they register with the user-led matching server using a pseudonymous (e.g., their email address, or their Reddit user ID) in order to commit to each other. Alice issues a commitment to Bob that indicate her willingness to be matched with Bob, if and only if Bob also (independently) commits to Alice. With this protocol, un-matched commitments only reveal who issued them, and not who they are issued for.

There are a number of problems that are related to privacy-preserving matching, such as secret handshakes [3], private set intersection [14] (and *cardinality-only* private set intersection [7]), trust negotiation [4], hidden credentials [17], oblivious envelopes [19], and policy-based encryption [1]. However, none of these tools can satisfactory address this problem. We elaborate on this in Section 2.

Contributions. We formalize the notions of privacy-preserving user-led and server-led matching. We then introduce two protocols which implement these functionalities. Our first protocol improves on existing work on secure user-led matching in terms of efficiency and simplicity. Our second protocol is, to our knowledge, the first to implement secure server-led matching. This is an important security primitive, because its non-private counterpart (i.e., a trusted server

that performs this functionality) is widely used in services such as Tinder [25], Hitch [16], and Cuddlr [8]. Furthermore, Facebook [12], Twitter [28] and Linked-In [20] offer user matching as part of their services.

Although our protocols are presented with two users, they naturally extend to any number of users; in the rest of the paper, Alice and Bob are two arbitrary users from the set of all users. Since existing social networks have millions (or even *billions* [13]) of users, practical privacy-preserving matching protocols must be very efficient. For this reason, all functionalities of our protocols are designed to require constant (i.e., $\mathcal{O}(1)$) computation on all parties. Moreover, our techniques do not require users who are not involved in a specific match to participate in the protocol.

Our protocols are formally proven secure in our framework. Unlike previous work [2, 21, 30], we relax the requirement for anonymous communication channels. Removing this requirement rules out any direct communication between the two parties, since it would allow them to identify each other using artifacts of the underlying communication channel, such as IP addresses.

Organization. The rest of the paper is organized as follows. Section 2 reviews the most prominent problems related to privacy-preserving matching, and discusses why they do not address it; the same section then discusses related work. Section 3 formally presents functionality and security properties of privacy-preserving matching. Protocol instantiations are introduced in Section 4, while their security analysis is addressed in Section 5. Section 6 presents extensions to our protocols. We conclude in Section 7.

2. RELATED WORK AND UNRELATED PROBLEMS

In this section, we review the most prominent related problems, and briefly discuss their relationship with privacy-preserving server-led and user-led matching. We follow this with a review of related work.

2.1 (Un-) Related Problems

As noted by Shin and Gligor in [24], there are a variety of problems that are somewhat related to privacy-preserving user matching, and for which secure and efficient protocols are known. Direct application of secure protocols for these problems in our setting, however, is not satisfactory for either security or efficiency reasons.

Secret handshakes. Secret handshakes allow two mutually untrusted parties to securely determine if they belong to the same group [3, 5, 9, 27]. There are clearly similarities between privacy-preserving matching and secret handshakes. However, protocols for secret handshake require either a shared secret between parties who belong to the same group, or the use of a certification authority. Neither is required in our server-led and user-led protocols. Our protocols are therefore suitable when the parties have no prior knowledge of each other.

Trust Negotiation. Clients can perform authentication without revealing the full extent of their credentials through trust negotiation protocols [4]. In principle, these protocols can be used to address privacy-preserving matching. However, analogously to PSI based instantiations, the resulting constructions will be inefficient, less secure and less practical.

Other Related Problems. Shin and Gligor in [23] mention several problems that are related to our setting, such as hidden credentials [17], oblivious envelopes [19] and policy-based encryption [1]. However, as noted in [23] these techniques focus mainly on the privacy of entities' attributes rather than on users' identity. Using these tools to construct user-led matching, if Alice commits to Bob she would also reveal her identity to him. This clearly does not satisfy our privacy requirement.

2.2 Related Work

Private Matchmaking. Baldwin and Gramlich defined the problem of Anonymous Matching in [2]. The paper addresses a problem that roughly corresponds to our user-chosen scenario, albeit not very efficiently. The paper proposes that "jokers" add fake transactions to the communication channel to "hide" real transactions. Baldwin and Gramlich's protocol requires a third party M to do the matching, and yet another party that the users trust to issue keys. Users exchange keys via M, who does not learn the key but *does* learn that the users are communicating. Fake transactions are supposed to make it more difficult for the server to detect actual users. The protocol was found to be vulnerable to *message replacement attack* by Zhang and Needham [30].

Meadows proposed an improvement [21] to the protocol by Baldwin and Gramlich. The protocol still requires a trusted third party which must be online when the users sign up for the system. The scenario presented in [21] consists of two representatives from two different companies that trust each other. Key exchange is handled between the companies, and focus is on immediate—rather then eventual—communication. The protocol in [21] is based on DH and assumes the two parties each have a secret. The objective is to check if the two secrets are the same without revealing them to each other.

In 2008, and later in 2013 Shin and Gligor [23, 24] revisited the problem of match-making protocols and added resistance to off-line guessing attacks and forward secrecy to the list of properties. Their protocol is based on secure password-based authenticated key exchange (PAKE). Although they provide a viable technique for user-led matching, their approach requires anonymous channels and does not address server-led matching.

Xie and Hengartner investigated in 2011 the problem of finding matches that fulfill some criteria, such as having shared interests [29]. The work is done in the context of mobile social networking, and uses a variant of private set intersection to achieve its goals. Xie and Hengartner's protocol can be compared with our user-led matching protocol, though it is significantly less efficient. It is not clear whether it is possible to construct a server-led matching protocol using the technique in [29], because there is no server involved in the protocol.

Private Set Intersection. Private set intersection (PSI) [14] allows two parties to determine which subset of elements appear in both their input sets, without revealing additional information. PSI can be used to construct secure privacy-preserving matching, albeit inefficiently (in the user-led case) or only with only honest-but-curious users (in the server-led case). Next, we briefly discuss how to use PSI to implement user-led and server-led privacy-preserving matching.

To implement user-led privacy-preserving matching, Alice commits to Bob, by adding element $\langle Alice, Bob \rangle$ to her set, where the two

elements are sorted, e.g., using lexicographic order. All users then engage in a separate instance of PSI with each user in the system. The output of Alice and Bob's PSI instance is $\langle Alice, Bob \rangle$ if and only if they committed to each other (i.e., both their sets contain $\langle Alice, Bob \rangle$). The cost of determining new matches is therefore $\mathcal{O}(n)$ instances of PSI for each user. The server must therefore mediate $\mathcal{O}(n^2)$ PSI instances. It is an open problem whether PSI can be used to implement user-led matching with cost $\mathcal{O}(1)$ for the users.

The use of PSI looks more promising for server-led matching. However the resulting protocol, discussed next, is inefficient, and is secure only in presence of semi-honest users. The server asks Alice and Bob if they would like to match. If Alice wishes to match with Bob, she creates a set containing an agreed-upon public constant. Otherwise, her set contains one random value. Bob builds his set analogously. Then Alice and Bob engage in PSI, using the server to forward protocol messages in order to conceal their real identity, and learn if both their sets contain the public constant. Let Alice be the party that learns the protocol's output. A malicious Alice could commit to Bob, and then arbitrarily replace the protocol's output, hence learning Bob's commitment without necessarily committing to him. Furthermore, this approach requires multiple communication rounds between Alice and Bob.

Server-Assisted Private Set Intersection and Private Equality Test. Kamara et al. introduced server-assisted PSI (SAPSI) in [18]. With SAPSI, two or more clients agree on a shared secret key k, and independently compute a PRP (keyed with k) over the elements of their input sets. The result of the permutations is sent to the server, which computes the intersection of these values and returns it to the clients. Because k is not known to the server, and the server does not collude with any other party, the intersection (and nothing else) is disclosed exclusively to the clients. Although the protocol is very efficient (communication and computation complexity is linear in the sum of the number of elements in the clients' sets, and all operations can be implemented using symmetric cryptography), we argue that it is not suitable for implementing user-led and server-led privacy-preserving matching. When used for user-led matching, the protocol assumes that all users share a secret key k, while the server has no access to k. This is clearly not possible in practice. Kamara et al. also present a technique for implementing server-aided private equality test. Although their protocol can be used to implement server-led privacy-preserving matching, it requires a substantially larger exchange of messages between the parties than our approach (due to the dummies used in the protocol). Moreover, it involves several rounds of communication between the protocol participants. If the parties are not on-line at the same time, this can possibly lead to substantial delays in the execution of the protocol.

3. SERVER-LED AND USER-LED MATCHING

In this section we formally define the functionality and security of our server-led and user-led matching protocols. SystemSetup algorithm is part of both protocols, and generates the system parameters which are subsequently made publicly available to all parties. This step is performed only once.

3.1 Protocol Functionality

Server-Led Matching Protocol. A server-led matching protocol is defined by a tuple of efficient algorithms (SystemSetup, UserSetup, Challenge, Response, Verify, Commit, Combine, Open), described below, and combined in three phases: Registration, Query and Notify, illustrated in Figure 1.

In the Registration phase, Alice generates a public-private pair of values using UserSetup. She then sends the public value to the server, which challenges her to prove that she knows the corresponding private value. During the Query phase, the server sends Bob's public parameter and description to Alice, who issues a commitment computed using the Commit algorithm, on input her decision, private parameter and Bob's public parameter. (In practice, the server would concurrently perform the Query phase with Alice and Bob.) Once the server receives both Alice and Bob's commitments, it combines them using the Combine algorithm. Finally, in the Notify phase, the server returns the combined commitment to the users, who learn the resulting decision after invoking the Open algorithm. SystemSetup, UserSetup, Commit, Combine and Open are formally defined next:

DEFINITION 1. *A server-led matching protocol involves the following efficient algorithms:*

- SystemSetup*: a probabilistic algorithm that, on input a security parameter 1^κ, outputs a public system parameter Φ.*
- UserSetup*: a probabilistic algorithm that takes as input Φ and outputs a secret parameter x_A and a public parameter y_A for Alice.*
- Challenge*: a probabilistic algorithm that takes as input y_A, and outputs a challenge γ and a (secret) response v.*
- Response*: a probabilistic algorithm that takes as input γ and x_A, and outputs a response σ_A.*
- Verify*: a probabilistic algorithm that takes as input σ_A and v, and outputs 1 iff the σ_A is a correct response.*
- Commit*: a probabilistic algorithm that takes as input Alice's decision $d_A \in \{\mathsf{yes}, \mathsf{no}\}$, x_A, Bob's public parameter y_B, and outputs a commitment c_A.*
- Combine*: a probabilistic algorithm that takes as input two commitments c_A, c_B and outputs a combined commitment c_{AB}. If $d_A = d_B =$ yes, then c_{AB} is a* yes *commitment, otherwise c_{AB} is a* no *commitment.*
- Open*: a deterministic algorithm that takes as input c_A, x_A, y_B, and outputs* yes *if c_{AB} is a* yes *commitment, and* no *otherwise.*

User-Led Matching Protocol. A user-led matching protocol is defined by a tuple of efficient algorithms (SystemSetup, UserSetup, Challenge, Response, Verify, Commit, Match, Open). These algorithms are combined in four phases (Registration, Lookup, Commit and Check) as illustrated in Figure 2. Similarly to the server-led matching protocol, during the Registration phase Alice generates a public-private pair of values using UserSetup. She then sends the public value y_A to the server, which challenges her to prove knowledge of the private value x_A. In the lookup phase, Alice requests the public parameter of a user of her choice—say, Bob. If Bob has registered, the server returns his public parameter y_B. Otherwise, it returns a properly distributed random value. During the Commit phase, Alice decides to commit to Bob. To do so, she computes a commitment c_A using her own private value x_A and the public value of Bob y_B. The server stores c_A together with Alice's identity. Finally, in the Check phase, Alice sends a "Check" message to the server, which computes the set of values $\mathcal{M}$ that

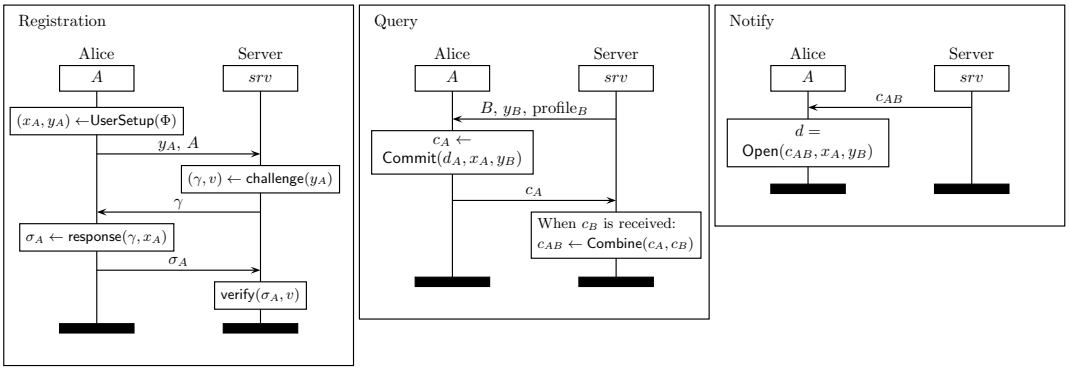

Figure 1: Server-led matching protocol. The protocol is composed of three phases, "Registration", "Query" and "Notify", each of which can be executed independently. The semantic relationship between the phases is illustrated in Figure 3.

represents matching commitments for Alice. Alice uses the Open algorithm on $\mathcal{M}$ to determine which users, to whom she committed, also committed to her.

SystemSetup, UserSetup, Challenge, Response, Verify, Commit, Match and Open are presented in Definition 2. We indicate the set of all users with $\mathcal{U}$, and the set of all commitments sent to the server as $\mathcal{D}$.

DEFINITION 2. *A user-led matching protocol consists of the following efficient algorithms:*

- SystemSetup*: a probabilistic algorithm that takes a security parameter 1^κ, and outputs a public system parameter Φ.*
- UserSetup*: a probabilistic algorithm that takes as input Φ and outputs a secret parameter x_A and a public parameter y_A for Alice.*
- Challenge*: a probabilistic algorithm that takes as input y_A, and outputs a challenge γ and a (secret) response v.*
- Response*: a probabilistic algorithm that takes as input γ and x_A, and outputs a response σ_A.*
- Verify*: a probabilistic algorithm that takes as input σ_A and v, and outputs 1 iff the σ_A is a correct response.*
- Commit*: a probabilistic algorithm that takes as input x_A, Bob's public parameter y_B, and outputs a commitment c_{AB}*
- Match*: a deterministic algorithm that takes as input the set of all commitments from Alice, defined as:*

$$\mathcal{C} \triangleq \left\{c_{Ai} \mid c_{Ai} \in \mathcal{D} \ \wedge \ c_{Ai} \leftarrow \mathsf{Commit}(x_A, y_i)\right\}_{i \in \mathcal{U}}$$

and a set of commitments issued by all users except for Alice:

$$\mathcal{S} \triangleq \left\{c_{Aj} \mid c_{Aj} \in \mathcal{D} \ \wedge \ c_{Aj} \leftarrow \mathsf{Commit}(x_j, y_A)\right\}_{j \in (\mathcal{U}')}$$
$$\cup \left\{c_{lj} \mid c_{lj} \in \mathcal{D} \ \wedge \ c_{lj} \leftarrow \mathsf{Commit}(x_l, y_j)\right\}_{l,j \in (\mathcal{U}')}$$

where $\mathcal{U}' = \mathcal{U} \setminus \{Alice\}$. This algorithm returns a set $\mathcal{M} = \{w_i \mid (c_{Ai} \in \mathcal{C}) \wedge (c_{Aj} \in \mathcal{S}) \wedge (i = j) \wedge (w_i = f(c_{Ai}, c_{Aj}))\}$ for some function f.
- Open*: a deterministic algorithm that takes as input $\mathcal{M}$, x_A, $\{y_i\}$, and outputs the set of users $\{i \mid w_i \in \mathcal{M}\}$.*

3.2 Security Properties of Matching Protocols

Both server-led and user-led matching protocols require a registration phase during which Alice proves that she is the owner of the public key being registered. Therefore, all security definitions in which $\mathcal{A}$ acts as the server assume that all users have registered their public parameters with $\mathcal{A}$.

Server-Led Matching. A secure server-led matching protocol must prevent the server from learning any information about the commitment issued by Alice and Bob, i.e., the server should learn neither the content of the individual commitments, nor the result of a match. To formally define this notion, we introduce an experiment where the server is allowed to provide Alice and Bob with two pairs of decisions $(d_{A,0}, d_{B,0})$ and $(d_{A,1}, d_{B,1})$ of its choice (e.g., (yes, no) and (no, no)). If the protocol reveals no information about the individual decisions, or about the resulting output, then the server cannot determine whether Alice and Bob selected $(d_{A,0}, d_{B,0})$ or $(d_{A,1}, d_{B,1})$ as their inputs. We formalize this notion as *indistinguishability under chosen commitments attack* (IND-CxA):

Experiment $\mathsf{IND\text{-}CxA}_{\mathcal{A}}(\kappa)$

1. $\mathcal{A}$ participates in the server-led matching protocol as the server, and interacts with Alice and Bob (both honest users). $\mathcal{A}$ is provided with the public parameters of the users via the registration phase which is run by Alice and Bob with $\mathcal{A}$.
2. $\mathcal{A}$ selects two pairs of decisions $(d_{A,0}, d_{B,0})$ and $(d_{A,1}, d_{B,1})$, where $d_{u,\beta} \in \{\mathsf{yes}, \mathsf{no}\}$ ($u \in \{A, B\}$ and $\beta \in \{0, 1\}$), and sends them to the honest users.
3. Alice and Bob select a common random bit b and run the protocol using $d_{A,b}$ and $d_{B,b}$ as their respective decisions.
4. At the end of the protocol execution, $\mathcal{A}$ outputs b' as its guess for b. The experiment outputs 1 if $b' = b$, and 0 otherwise.

DEFINITION 3 (IND-CxA SECURITY). *A server-led matching protocol has indistinguishable commitments under chosen commitment attack if there exists a negligible function* negl *such that for any PPT $\mathcal{A}$, $\Pr[\mathsf{IND\text{-}CxA}_{\mathcal{A}}(\kappa) = 1] \leq 1/2 + \mathsf{negl}(\kappa)$.*

Another important security property of a server-led matching protocol is the inability of Alice to learn information about Bob's decision when she issues a no commitment. (If Alice issues a yes commitment, she always learns Bob's decision.) This property is captured by *negative commitment attack* (IND-NCA) security:

Experiment $\mathsf{IND\text{-}NCA}_{\mathcal{A}}(\kappa)$

1. $\mathcal{A}$ participates in the protocol as a user, and interacts with an honest server and an honest user (Bob).

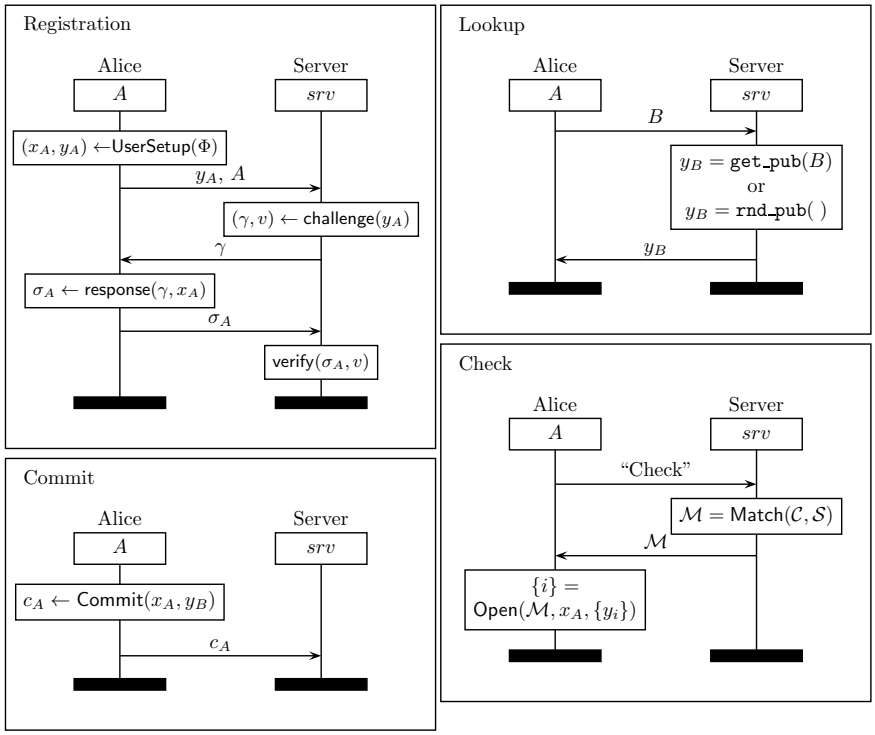

Figure 2: User-led matching protocol. The protocol is composed of four phases, "Registration", "Lookup", "Commit" and "Check", each of which can be executed independently. The semantic relationship between the phases is illustrated in Figure 4.

2. $\mathcal{A}$ issues a commitment, and Bob selects a random bit b. If $b = 0$, then Bob issues a yes commitment; otherwise, he issues a no commitment.
3. At the end of the protocol execution, $\mathcal{A}$ outputs b' as its guess for b. The experiment outputs 1 if $b' = b$ and $\mathcal{A}$'s commitment is no, and 0 otherwise.

DEFINITION 4 (IND-NCA SECURITY). *A server-led matching protocol has indistinguishable commitments under negative commitment attack if there exists a negligible function* negl *such that for any PPT* $\mathcal{A}$, $\Pr[\mathsf{IND\text{-}NCA}_{\mathcal{A}}(\kappa) = 1] \leq 1/2 + \mathsf{negl}(\kappa)$.

User-Led Matching. In a secure user-led matching protocol, the server must not be able to determine which user Alice has committed to, unless there already exists a corresponding commitment from that user to Alice. To represent this, we allow Alice to commit to one of two users U_0 and U_1, who did not commit to her, and challenge the adversary to determine who Alice committed to. This notion is captured by the server matching attack (IND-SMA) experiment.

Experiment $\mathsf{IND\text{-}SMA}_{\mathcal{A}}(\kappa)$
1. $\mathcal{A}$ participates in the protocol as the server, and interacts with an honest user Alice.
2. $\mathcal{A}$ performs registration with Alice and two users, U_0 and U_1.
3. Alice is given two sets of public parameters, corresponding to U_0 and U_1.
4. Alice selects a random bit b, commits to U_b and sends the commitment to $\mathcal{A}$.
5. At the end of the protocol execution, $\mathcal{A}$ outputs b' as its guess for b. The experiments outputs 1 if $b' = b$, and 0 otherwise.

DEFINITION 5 (IND-SMA SECURITY). *A user-led matching protocol has indistinguishable commitments under server-matching attack if there exists a negligible function* negl *such that for any PPT* $\mathcal{A}$, $\Pr[\mathsf{IND\text{-}SMA}_{\mathcal{A}}(\kappa) = 1] \leq 1/2 + \mathsf{negl}(\kappa)$.

A secure user-led matching protocol must prevent a malicious user from forging commitments from a different user. This notion is captured by our forging commitment attack experiment (FCA), presented next. In our protocol instantiation (see Section 4), IND-SMA security implies FCA security. However in general this is not true.

Experiment $\mathsf{FCA}_{\mathcal{A}}(\kappa)$
1. $\mathcal{A}$ participates in the user-led matching protocol as a user, and interacts with an honest server.
2. $\mathcal{A}$ performs the lookup protocol multiple times, obtaining public parameters of users $U_1, \ldots, U_j$ as well as their corresponding secret information.
3. Eventually, $\mathcal{A}$ is given the public parameter of two new users U_A and U_B and outputs a value C.
4. The experiment outputs 1 if C is a valid commitment that binds U_A and U_B; otherwise it outputs 0.

DEFINITION 6 (FCA SECURITY). *A user-led matching protocol has unforgeable commitments under a forging commitment attack if there exists a negligible function* negl *such that for any PPT* $\mathcal{A}$, $\Pr[\mathsf{FCA}_{\mathcal{A}}(\kappa) = 1] \leq \mathsf{negl}(\kappa)$.

4. PROTOCOL INSTANTIATIONS

In this section we present our instantiation server-led and user-led matching.

4.1 Server-Led Matching Protocol

Our server-led matching protocol uses a collision-resistant hash function $\mathrm{H}(\cdot)$ and a semantically secure additively homomorphic encryption scheme: $\mathsf{Setup}(\cdot)$, $\mathsf{Enc}(\cdot)$, $\mathsf{Dec}(\cdot)$. In an additively homomorphic encryption scheme, $\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 + m_2)$, which also implies that $\mathsf{Enc}(m)^a = \mathsf{Enc}(a \cdot m)$. (Public/private keypairs are generated using $\mathsf{Setup}(\cdot)$, and it holds that $\mathsf{Dec}(\mathsf{Enc}(m)) = m$.) Additionally, our constructions require that the message space of the homomorphic encryption scheme used is a field. Any encryption scheme with the above properties, such as the Paillier [22] or DGK [10, 11], suffices for the purposes of this work. Next, we introduce the instantiations of each algorithm in Definition 1:

- $\mathsf{SystemSetup}(1^\kappa)$: Outputs $\Phi = (p, q, g)$, i.e., the description of a cyclic group $\mathbb{G}$, a generator g of a subgroup of size q of $\mathbb{G}$ and prime p s.t. $q|p-1$.
- $\mathsf{UserSetup}(\Phi)$: Picks a random value $x_A \leftarrow \mathbb{Z}_q$ and sets $y_A = g^{x_A}$. Then, it outputs (x_A, y_A).
- $\mathsf{Challenge}(y_A)$: Picks a random value $r \leftarrow \mathbb{Z}_q$ and sets $v = (y_A)^r$. Then, it outputs $\gamma = (g^r, \mathrm{H}(v))$.
- $\mathsf{Response}(\gamma)$: Computes $\sigma = (g^r)^{x_A}$. If $\mathrm{H}(v) \neq \mathrm{H}(\sigma)$, the algorithm aborts. Otherwise, it returns σ.
- $\mathsf{Verify}(\sigma, v)$: Returns 1 if $\sigma = v$, and 0 otherwise.
- $\mathsf{Commit}(d_A, x_A, y_B)$: Computes $(pk, sk) \leftarrow \mathsf{Setup}(1^\kappa, g^{x_A x_B})$. If $d_A = \mathsf{yes}$, then $m_A = 0$; otherwise, m_A is selected uniformly at random from the message space of Enc. The output is set to $c_A = \mathsf{Enc}(m_A)$.
- $\mathsf{Combine}(c_A, c_B)$: Selects a random element s from the message space and computes $c_{AB} = (c_A \cdot c_B)^s = \mathsf{Enc}((m_A + m_B) \cdot s)$. If $m_A = 0$ and $m_B = 0$, then c_{AB} is the encryption of 0.[1] Otherwise, c_{AB} corresponds to the encryption of a random element from the message space of Enc.
- $\mathsf{Open}(c_{AB})$: Computes $(pk, sk) \leftarrow \mathsf{Setup}(1^\kappa, r)$ where $r = (y_B)^{x_A} = g^{x_A x_B}$ and sets $d_{AB} = \mathsf{Dec}(c_{AB})$. If $d_{AB} = 0$, then it outputs yes; otherwise, it outputs no.

A possible execution of our server-led protocol is illustrated in Figure 3. For clarity, we include both Alice and Bob. However, the protocol is independently executed by each user, and with no timing constraints on messages.

The purpose of the challenge/response/verify steps, executed during the registration phase, is to verify that the new user has knowledge of the secret key associated with the public parameter she is registering. In other words, these steps guarantee that the user is not registering somebody else's public parameter under her name. The steps also guarantee that the server does not learn any additional information that can only be computed using Alice's private key. This is also important for Alice, as it would otherwise be possible for the server to force Alice to produce a valid commitment for Bob by replacing g^r with $y_B = g^{x_B}$. However since the server does not know x_B, it cannot compute $\mathrm{H}((g^r)^{x_B})$. A formal argument for the security of these steps is provided as proof of Theorem 5 in Appendix A. To simplify exposition, we omit treatment of the registration phase from the security proofs presented in Section 5.

4.2 User-Led Matching Protocol

In this Section we present our instantiation of user-led matching. Our protocol uses a secure pseudo-random function $\mathrm{PRF}_k(\cdot)$, keyed with Alice and Bob's shared secret, to generate their commitments. This allows us to generate additional keys, known only to Alice and Bob, as discussed below. The keys are used to securely exchange information between the users through the Server.

The registration phase of the user-led matching protocol is the same as in the server-led protocol. However for the sake of completeness we present the instantiation of all algorithms, including $\mathsf{SystemSetup}$ and the four algorithms used in the Registration phase: $\mathsf{UserSetup}$, $\mathsf{Challenge}$, $\mathsf{Response}$, and Verify.

- $\mathsf{SystemSetup}(1^\kappa)$: Outputs $\Phi = (p, q, g)$, i.e., the description of a cyclic group $\mathbb{G}$, a generator g of a subgroup of size q of $\mathbb{G}$ and prime p s.t. $q|p-1$. (Identical to $\mathsf{SystemSetup}$ of Server-Led protocol.)
- $\mathsf{UserSetup}(\Phi)$: Picks a random value $x_A \leftarrow \mathbb{Z}_q$ and sets $y_A = g^{x_A}$. Then, it outputs (x_A, y_A). (Identical to $\mathsf{UserSetup}$ of Server-Led protocol.)
- $\mathsf{Challenge}(y_A)$: Picks a random value $r \leftarrow \mathbb{Z}_q$ and sets $v = (y_A)^r$. Then, it outputs $\gamma = (g^r, \mathrm{H}(v))$. (Identical to $\mathsf{Challenge}$ of Server-Led protocol.)
- $\mathsf{Response}(\gamma)$: Computes $\sigma = (g^r)^{x_A}$. If $\mathrm{H}(v) \neq \mathrm{H}(\sigma)$, the algorithm aborts. Otherwise, it returns σ. (Identical to $\mathsf{Response}$ of Server-Led protocol.)
- $\mathsf{Verify}(\sigma, v)$: Returns 1 if $\sigma = v$, and 0 otherwise. (Identical to Verify of Server-Led protocol.)
- $\mathsf{Commit}(x_A, y_B)$: Sets $k_A = (y_B)^{x_A} = g^{x_A x_B}$ and outputs $c_A = \mathrm{PRF}_{k_A}(0)$, where PRF is a pseudorandom function.
- $\mathsf{Match}(\mathcal{C}, \mathcal{S})$: Outputs $\mathcal{M} = \mathcal{C} \cap \mathcal{S}$, where $\mathcal{C}$ is the set of all commitments from Alice and $\mathcal{S}$ is the set of all commitments from all users except for Alice. (Since $k_A = (y_B)^{x_A} = (y_A)^{x_B} = g^{x_A x_B} = k_B$, if Alice commits to Bob and Bob commits to Alice, then $\mathrm{PRF}_{k_A}(0)$ appears in both $\mathcal{C}$ and $\mathcal{S}$.)
- $\mathsf{Open}(\mathcal{M}, x_A, \{y_i\})$: Outputs all users i for which $\mathrm{PRF}_{(y_i^{x_A})}(0) \in \mathcal{M}$. (In other words, given a commitment from Alice to Bob, outputs "Bob").

Figure 4 represents a possible execution of our user-led protocol. During the Registration phase, the server checks if Alice has knowledge of x_A as discussed in the instantiation of the server-led matching protocol. The Check phase of the protocol will only return matches if Bob has also executed the protocol with the server, and has committed to Alice. Without loss of generality, in the rest of the paper we assume that Check is executed after each Commit, and that therefore the server only checks if the *new* commitment matches any *existing* commitment.

As an extension not show in Figure 4, Alice can optionally compute $k' = \mathrm{PRF}_{k_A}(1)$. k' is then used to encrypt an additional message msg_A as $w_A = E_{k'}(msg_A)$ using a symmetric encryption scheme during the Commit phase. w_A is then sent to the server along with c_A. If Bob commits to Alice, the server returns w_A to Bob (and possibly the corresponding w_B to Alice) during the Check phase. Both Alice and Bob can decrypt w_A and w_B since they can reconstruct k' from $g^{x_A x_B}$.

Retrieving Public Parameter *Privately*. In our user-led matching protocol, Alice must retrieve Bob's public parameter before committing to him. Although knowing that Alice obtained Bob's public parameter does not prove that she will commit to him, this still represent information leakage. This problem can be mitigated if each new user could be provided with a complete list of all existing users after registering. This way, the server learns nothing. Although this

[1] Note that Enc is semantically secure and thus the adversary cannot tell if two ciphertexts encrypt the same value.

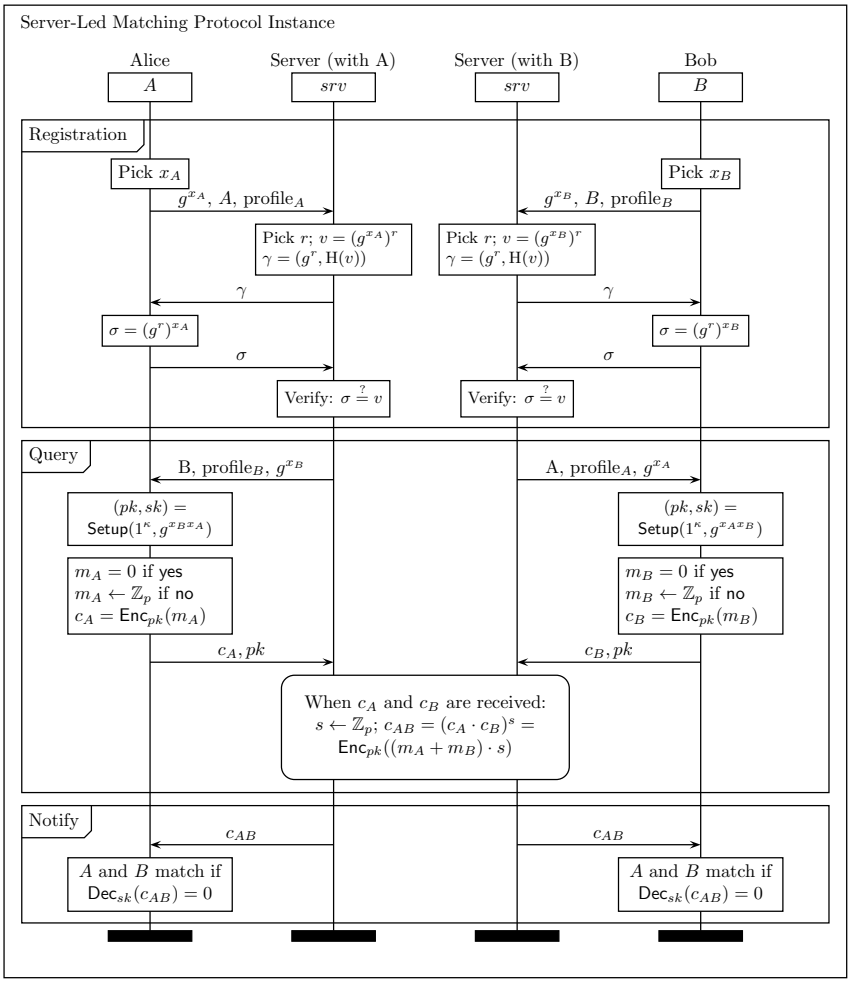

Figure 3: Example of server-led matching protocol instance, where the protocol phases (see Figure 1) are executed in the following order: Registration, Query, and Notify. The server separately executes the entire protocol with both Alice and Bob.

might work for a small number of users (the keys and identifiers of 100,000 users occupy 20-30 MB), it does not scale to *millions* of users. Moreover, this discloses the complete list of users enrolled in the system, which in itself might be sensitive.

A more efficient approach is to use a private information retrieval (PIR) protocol [6], which allows Alice to privately request arbitrary identities from the server. As an alternative, assuming that the volume of requests is sufficiently high, Alice can issue anonymous requests to the server via an anonymizing service such as Tor [26].

5. PROTOCOL ANALYSIS

In this section we analyze the correctness, efficiency and security of our server-led and user-led protocol instantiations. In our security analysis, we use the term *adversary* to refer to insiders, i.e., protocol participants. External adversaries are not considered, since their actions can be mitigated via standard network security techniques, i.e., confidential and authenticated channels. Our protocols are secure in the presence of *malicious* users and *semi-honest* (also known as honest-but-curious or passive) server. We refer the reader to [15] for the formal definitions of the malicious and semi-honest models. Although we continue our analysis of the server's behavior in the semi-honest model, it is important to note that if the server turns malicious, it can only break the *functionality* and *fairness* of the matchmaking protocol, not the privacy of the user's choices.

Our scheme requires the underlying homomorphic encryption scheme to be semantically secure. To formalize this notion, we use the standard IND-CPA security definition, reported next:

Experiment $\mathsf{IND\text{-}CPA}_{\mathcal{A},\mathcal{E}}(\kappa)$

1. Run $(pk, sk) \leftarrow \mathsf{Setup}(1^\kappa, r)$.
2. Adversary $\mathcal{A}$ is given pk and eventually outputs two messages m_0, m_1 of its choice.
3. A random bit b is drawn and the encryption $\mathsf{Enc}_{pk}(m_b)$ is returned to $\mathcal{A}$.
4. $\mathcal{A}$ outputs bit b', and the experiment outputs 1 iff $b = b'$.

DEFINITION 7 (IND-CPA SECURITY). *An encryption scheme* $\mathcal{E} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *has indistinguishable encryptions under chosen plaintext attack if there exists a negligible function* negl *such that for any probabilistic polynomial time (PPT)* $\mathcal{A}$, $\Pr[\mathsf{IND\text{-}CPA}_{\mathcal{A},\mathcal{E}}(\kappa) = 1] \leq 1/2 + \mathsf{negl}(\kappa)$.

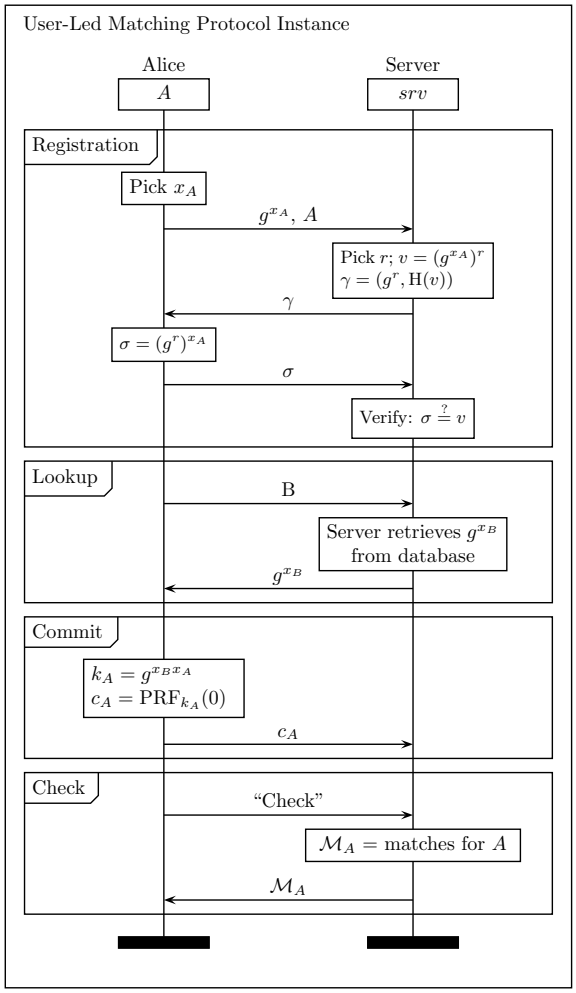

Figure 4: Example of user-led matching protocol instance, where the four phases (Figure 2) are executed in the following order: Registration, Lookup, Commit, and Check.

5.1 Server-Led Matching Protocol

Correctness. Alice and Bob independently run Commit on input (d_A, x_A, y_B) and (d_B, x_B, y_A) respectively. Since Setup is deterministic and its inputs $(y_B)^{x_A}$ from Alice and $(y_A)^{x_B}$ from Bob are identical, the two users compute the same keypair (pk, sk). If both Alice and Bob issue a yes commitment, i.e., $m_A = m_B = 0$, then $c_{AB} = (c_A \cdot c_B)^s = \mathsf{Enc}((0+0) \cdot s) = \mathsf{Enc}(0)$. On the other hand, if at least one of the participants issues a no commitment, c_{AB} is the encryption of a random element in the message space of Enc.

Computational Complexity. The server can implement Registration in constant time by storing user information in a hash table. The cost of the Query and Notify phases is also constant. The space required for the server to run the protocols grows linearly with the number of users due to the cost of storing their public parameters and additional information. Both time and space complexity for users are constant.

THEOREM 1. *Assuming that the underlying homomorphic encryption scheme is semantically secure and that DDH holds in* $\mathbb{G}$*, our server-led matching protocol is* IND-CxA*-secure against a semi-honest server.*

PROOF. We show that an adversary $\mathcal{A}$ that has non-negligible advantage (which we indicate as $\delta(\cdot)$) over $1/2$ to win the server matching privacy experiment can be used to break the semantic security of the homomorphic encryption scheme used to instantiate the protocol. Let SIM_S be a simulator that interacts with the server as follows. SIM_S plays the IND-CPA experiment and receives pk from the IND-CPA challenger. It generates two random values x_A and x_B and sends pk, g^{x_A} and g^{x_B} to $\mathcal{A}$. SIM_S then receives $(d_{A,0}, d_{B,0})$ and $(d_{A,1}, d_{B,1})$ from $\mathcal{A}$. It then sends $ch_u = (m_{u,0}, m_{u,1})$ to the IND-CPA challenger, where $u \leftarrow \{A, B\}$ and $m_{u,\beta} = 0$ if $d_{u,\beta} = 0$, and $m_{u,\beta}$ is a random value otherwise. The IND-CPA challenger returns $\mathsf{Enc}_{pk}(m_{u,b})$. SIM_S sends $\{(\mathsf{Enc}_{pk}(m_{u,b}), x), (\mathsf{Enc}_{pk}(m_{\bar{u},b''}), \bar{u})\}$ where $\bar{u} \in \{A, B\} \setminus \{u\}$ (i.e., if u is A then $\bar{u}$ is B, and vice-versa) and b'' is a random bit. $\mathcal{A}$ might abort because pk has not been generated using g^{u_A} and g^{u_B} as input of $Setup$. However, if this is the case, it is easy to see that this behavior can be used to break DDH in $\mathbb{G}$ as follows. Given a DDH challenge $(g, g^{x_1}, g^{x_2}, g^{x_3})$, compute $(pk', sk') = Setup(1^\kappa, g^{x_3})$ and send pk, g^{x_1} and g^{x_2} to $\mathcal{A}$. We have that, with non-negligible advantage over 1/2, $x_3 \neq x_1 \cdot x_2$ iff $\mathcal{A}$ aborts. Since this violates the assumption that DDH is hard in $\mathbb{G}$, $\mathcal{A}$ cannot determine whether pk was generated using g^{x_1} and g^{x_2}.

Eventually $\mathcal{A}$ outputs b'; SIM_S outputs the same value. If $b'' = b$, then $\{(\mathsf{Enc}_{pk}(m_{u,b}), u), (\mathsf{Enc}_{pk}(m_{\bar{u},b''}), \bar{u})\}$ is a proper challenge for $\mathcal{A}$, and therefore $b'' = b$ with probability $1/2 + \delta(\kappa)$. Otherwise, the challenge reveals no information about b and therefore $\mathcal{A}$ can guess correctly with probability $1/2$. Therefore, we have that $b' = b$ with probability $1/2 \cdot (1/2 + \delta(\kappa) + 1/2 \cdot 1/2) = 1/2 + 1/2 \cdot \delta(\kappa)$. Since $\delta(\cdot)$ is non-negligible, also $1/2 \cdot \delta(\cdot)$ is non-negligible. This contradicts the semantic security of the homomorphic encryption scheme. □

THEOREM 2. *Our server-led matching protocol is* IND-NCA*-secure against a malicious user.*

PROOF. After interacting with the honest server, $\mathcal{A}$ learns $(y + m_b) \cdot s$. Since $y + m_b$ is not equal to zero and s is uniformly distributed in the message space, $(y + m_b) \cdot s$ does not reveal any information about m_b. □

5.2 User-Led Matching Protocol

Correctness. Alice commits to Bob by computing $c_A = \mathrm{PRF}_{k_A}(0)$, while Bob commits to Alice with $c_B = \mathrm{PRF}_{k_B}(0)$. We have that $k_A = (y_B)^{x_A} = (g^{x_B})^{x_A} = g^{x_A x_B} = (g^{x_A})^{x_B} = (y_A)^{x_B} = k_B$. Since PRF is deterministic, $\mathrm{PRF}_{k_A}(0) = \mathrm{PRF}_{k_B}(0)$ and therefore $c_A = c_B$.

Computational Complexity. The protocol is efficient for both the server and the users. The server can implement Registration, Lookup and Commit in constant time, and Check in linear time in the number of commitments from Alice by storing user information and commitments in two separate hash tables. (In practice, the number of commitments issued by Alice is negligible compared to the number of users.) The space required for the server to run the protocols grows linearly with the number of users and the number of commitments issued by the users. Both time and space complexity for users are constant.

THEOREM 3. *Assuming that DDH holds in $\mathbb{G}$, our user-led matching protocol is* IND-SMA*-secure against a semi-honest server.*

PROOF. We show that a simulator SIM_U can use $\mathcal{A}$ to break DDH in $\mathbb{G}$ as follows. Let $(g, g^{x_1}, g^{x_2}, g^{x_3})$ be a DDH challenge, i.e., $P[x_3 = x_1x_2] = 1/2$. SIM_U sets the system's public parameter to $\Phi = (g, p, q)$, its public parameter to g^{x_1}, the public parameter of U_β to g^{x_2} and the public parameter of $U_{\neg\beta}$ to a random element $r \leftarrow \mathbb{G}$ and its commitment to $c = \mathrm{PRF}_{g^{x_3}}(0)$ (β is a random bit). Then, it sends $(g, g^{x_1}, g^{x_2}, r, c)$ to $\mathcal{A}$.

$\mathcal{A}$ might abort because if $g^{x_3} \neq g^{x_1x_2}$, then c is not a legitimate commitment of SIM_U with either U_0 or U_1. However, if this is the case, it is easy to see that this behavior can be used to break DDH in $\mathbb{G}$, since $\mathcal{A}$ would abort if $(g, g^{x_1}, g^{x_2}, g^{x_3})$ is not a DDH tuple, and output b' otherwise.

Eventually, $\mathcal{A}$ outputs its choice b'. SIM_U outputs 1 if $b' \neq \beta$, and $a \leftarrow \{0, 1\}$ otherwise. It is easy to see that when $b' \neq \beta$, SIM_U's answer is correct iff $\mathcal{A}$'s answer is correct. Therefore, SIM_U's output is correct with probability $1/2 + 1/4$. This violates the assumption that DDH is hard in $\mathbb{G}$, and therefore $\mathcal{A}$ cannot exist. □

THEOREM 4. *Assuming that DDH holds in $\mathbb{G}$, our server-led matching protocol is* FCA*-secure against a malicious user.*

PROOF. We show how a simulator SIM can interact with $\mathcal{A}$ to break DDH in $\mathbb{G}$. SIM receives a DDH challenge $(g, g^{x_1}, g^{x_2}, g^{x_3})$ and sets the system parameter to g, U_A's public parameter to g^{x_1} and U_B's public parameter to g^{x_2}. $\mathcal{A}$ eventually outputs $C = \mathrm{PRF}_{(g^{x_1x_2})}(0)$. Because C is equal to $\mathrm{PRF}_{(g^{x_3})}(0)$ with negligible probability if $x_3 \neq x_1x_2$, we have that $x_3 = x_1x_2$ with overwhelming probability. Therefore $(g, g^{x_1}, g^{x_2}, g^{x_3})$ is a DDH tuple with overwhelming probability if $C = \mathrm{PRF}_{(g^{x_3})}(0)$. Since DDH is hard in $\mathbb{G}$, $\mathcal{A}$ cannot output win the IND-CMA experiment with non-negligible probability. □

6. PROTOCOL EXTENSIONS

In this section we present protocol extensions that have been left out of the explanation of the basic protocols for clarity.

Committing to Users *and* Keywords. In the user-led matching protocol, instead of simply committing to Bob, Alice may want to specify one or more *keywords* that must be matched by both users, thus only committing to Bob if he also commits to keyword kw. In this case, Alice generates her commitment as $c_A = g^{x_A x_B \mathrm{H}(kw)}$ where $\mathrm{H}(\cdot)$ is a cryptographic hash function. Bob's commitment will match only if he also includes $\mathrm{H}(kw)$ as part of his commitment.

Information Transfer. Users of the server-led matching protocol might want to exchange additional information in case they both issue a positive commitment, without relying on additional communication via the server after executing the protocol. For example, if Alice and Bob's identifiers are nicknames not tied to any other on-line identity, they will need to exchange email addresses in order to communicate outside of the matching system. This can be done as follows. Alice encrypts her identifier id_A using pk as $e_A = \mathsf{Enc}_{pk}(id_A)$. Similarly, Bob computes e_B. e_A and e_B are then sent to the server which, given two the two commitments c_A and c_B, sends $(e_A \cdot (c_A \cdot c_B)^{s'}) = \mathsf{Enc}(id_A + s' \cdot (m_A + m_B))$ to Bob for a random s'. (Analogously, $\mathsf{Enc}(id_B + (m_A + m_B) \cdot s')$ is sent to Alice.) If both users commit to each other, then $(m_A + m_B) \cdot s' = 0$ and therefore each party receives the other user's identifier. Otherwise, $(m_A + m_B) \cdot s'$ is uniformly distributed in the message space, and therefore user do not learn the other party's identifier.

6.1 Security Extensions

Next, we discuss how to prevent attacks carried out by adversaries that are *outside the adversary model* considered thus far.

In both protocols a malicious server can claim that an honest user issued a commitment that she, in fact, did not generate. In the server-led matching protocol, after receiving a commitment from Alice, the server performs the matching computation with two copies of Alice's commitment, pretending that one of the copies came from Bob. (The server can correctly guess Bob's identity with low, albeit non-negligible, probability, given a commitment from Alice.) In the user-led protocol, the server can always return a false match. Analogously, after the server discloses a match to Alice and Bob, Bob might claim that he did not commit to Alice.

In other words, the protocols do allow the parties to prove their statements. However, the protocols can be easily modified to guarantee correctness by requiring users to sign their commitments using any public-key signature scheme. In case of a dispute, the server can reveal two commitments (one from Alice and one from Bob) with valid signatures if and only if it has faithfully followed the protocol. In the server-led matching protocol, after the server reveals both signed commitments, Alice decrypts Bob's commitment to verify the server's claim. (Similarly, Bob can verify Alice's claim.)

7. CONCLUSION

In this paper, we introduced a formal framework for securely matching two or more parties with related interests. We looked at two different scenarios: privacy-preserving *server-led* matching, where a server selects potential matching pairs, then collects and combines each party's *private* preference; and privacy-preserving *user-led* matching, where entities are allowed to express their private preference for any party.

We presented secure and efficient instantiations of both functionalities. Security of our protocols is supported by formal proofs, based on standard assumptions. Users are modeled as malicious entities, while the server is semi-honest. We also presented several extensions that enable the protocols to function (partially) with a fully malicious server.

Our protocols are practical and efficient. The cost of commitment generation and matching is constant, i.e., it does not depend on the number of users. Moreover, the cryptographic operations performed in the protocols are relatively inexpensive and the protocols involve only a few short messages per phase.

8. REFERENCES

[1] W. Bagga and R. Molva. Policy-based cryptography and applications. In *Financial Cryptography*, pages 72–87, 2005.

[2] R. Baldwin and W. Gramlich. Cryptographic protocol for trustable match making. *IEEE Security and Privacy Magazine*, 1985.

[3] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H. Wong. Secret handshakes from pairing-based key agreements. In *In IEEE Symposium on Security and Privacy*, pages 180–196, 2003.

[4] P. Bonatti and P. Samarati. Regulating service access and information release on the web. In *Proceedings of the 7th ACM Conference on*

Computer and Communications Security, CCS '00, pages 134–143, New York, NY, USA, 2000. ACM.
[5] C. Castelluccia, S. Jarecki, and G. Tsudik. Secret handshakes from ca-oblivious encryption. In *ASIACRYPT*, 2004.
[6] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
[7] E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In *CANS*, pages 218–231, 2012.
[8] Cuddlr. http://cuddlrapp.com/.
[9] Á. Cuevas, P. El Khoury, L. Gomez, A. Laube, and A. Sorniotti. A security pattern for untraceable secret handshakes. In *International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*, 2009.
[10] I. Damgård, M. Geisler, and M. Krøigård. A correction to efficient and secure comparison for on-line auctions. Cryptology ePrint Archive, Report 2008/321, 2008.
[11] I. Damgård, M. Geisler, and M. Krøigård. Homomorphic encryption and secure comparison. *Journal of Applied Cryptology*, 1(1):22–31, 2008.
[12] Facebook. http://www.facebook.com/.
[13] Facebook. Press Release. http://www.prnewswire.com/news-releases/facebook-reports-third-quarter-2013-results-229923821.html.
[14] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Eurocrypt*, pages 1–19. Springer-Verlag, 2004.
[15] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.
[16] Hitch. http://www.hitchapp.co/.
[17] J. Holt, R. Bradshaw, K. Seamons, and H. Orman. Hidden credentials. In *WPES*, pages 1–8. ACM, 2003.
[18] Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. Scaling private set intersection to billion-element sets. Technical Report MSR-TR-2013-63, June 2013.
[19] N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. *Distributed Computing*, 17(4):293–302, 2005.
[20] LinkedIn. http://www.linkedin.com/.
[21] C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE Symposium on Security and Privacy*, pages 134–137. IEEE Computer Society, 1986.
[22] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238, 1999.
[23] J. Shin and V. Gligor. A new privacy-enhanced matchmaking protocol. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2008.
[24] J. Shin and V. Gligor. A new privacy-enhanced matchmaking protocol. In *IEICE Transactions*, 2013.
[25] Tinder. http://www.gotinder.com/.
[26] Tor. https://www.torproject.org.
[27] G. Tsudik and S. Xu. A flexible framework for secret handshakes. In *Privacy Enhancing Technologies*, 2006.
[28] Twitter. http://www.twitter.com/.
[29] Qi Xie and U. Hengartner. Privacy-preserving matchmaking for mobile social networking secure against malicious users. In *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*, July 2011.
[30] K. Zhang and R. Needham. A private matchmaking protocol, 1998.

APPENDIX

A. CHALLENGE-RESPONSE-VERIFY STEPS IN THE REGISTRATION PHASE

In this section, we first show that the user can output a correct value σ for the *challenge-response-verify* steps in the Registration phase (in both protocols) for a given public parameter g^{x_A} only if she has knowledge of x_A, under the DDH assumption. Then, we prove that the server does not learn any new information, besides the fact that the user has knowledge of x_A, from the execution of the protocol.

More formally:

THEOREM 5. *Let $r \leftarrow \{0,1\}^\kappa$, $y = g^{x_A}$, $v = \mathrm{H}(y^r)$, and $\gamma = (g^r, \mathrm{H}(v))$. Assuming that DDH is hard in $\mathbb{G}$ and that $\mathrm{H}(\cdot)$ is a collision-resistant hash function, for all PPT algorithms C there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[C(y,\gamma) = g^{x_A \cdot r}] \leq \mathsf{negl}(\kappa)$.*

PROOF. Assume that there exists a PPT algorithm C that outputs $g^{x_A \cdot r}$ on input (y, γ) with some non-negligible probability indicated with $\delta(\kappa)$. We argue that the existence of C violates the assumption on the hardness of DDH in $\mathbb{G}$, since a simulator SIM_U can use C to answer a DDH challenge with probability $\delta(\kappa)$.

Let $ch = (g, g^{x_1}, g^{x_2}, g^{x_3})$ be a DDH challenge. SIM_U invokes $C(g^{x_1}, (g^{x_2}, \mathrm{H}(g^{x_3})))$. C can perform one of the following three actions: (1) it outputs $\sigma = g^{x_3}$; (2) it outputs $\sigma \neq g^{x_3}$; (3) it aborts. (SIM_U can trivially distinguish these three outcomes.) If C performs (1), then SIM_U outputs "ch is a DDH tuple". Otherwise, it outputs "ch is not a DDH tuple".

We argue that SIM_U's output is correct with probability $\delta(\kappa)$. In particular, action (1) must be performed by C with probability $\delta(\kappa)$ when $x_3 = x_1 \cdot x_2$, i.e., when ch is a DDH tuple. If ch is *not* a DDH tuple, then C cannot output $\sigma = g^{x_3}$ (except with negligible probability), because x_3 has been selected independently from x_1 and x_2. Therefore, C either performs (2) or (3) with overwhelming probability.

This is in violation of the assumption that DDH is hard in $\mathbb{G}$. Therefore, $\delta(\kappa) \leq \mathsf{negl}(\kappa)$. □

To show that the adversary learns no new information, besides the fact that the user knows the secret key associated with her public parameter, we now show that the server can construct a valid challenge only if it can compute the response to the challenge on its own, e.g., with knowledge of r.

THEOREM 6. *Let $r \leftarrow \{0,1\}^\kappa$ and $y = g^{x_A}$. Assuming that DDH is hard in $\mathbb{G}$, for all PPT algorithms S there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[S(y, g^r) = \mathrm{H}(y^r)] \leq \mathsf{negl}(\kappa)$.*

PROOF. Assume that there exists a PPT algorithm S that outputs $\mathrm{H}(y^r)$ on input (y, g^r) with some non-negligible probability $\delta(\kappa)$. We argue that the existence of S violates the assumption on the hardness of DDH, since a simulator SIM_S can use S to answer a DDH challenge with probability $\delta(\kappa)$.

Let $ch = (g, g^{x_1}, g^{x_2}, g^{x_3})$ be a DDH challenge. SIM_S invokes $S(g^{x_1}, g^{x_2})$. If S outputs $\mathrm{H}(g^{x_3})$, then SIM_S outputs "ch is a DDH tuple". Otherwise, it outputs "ch is not a DDH tuple".

We argue that SIM_S's output is correct with probability $\delta(\kappa)$. In particular, S outputs $\mathrm{H}(y^r)$ with probability $\delta(\kappa)$ when $x_3 = x_1 \cdot x_2$, i.e., when ch is a DDH tuple. If ch is *not* a DDH tuple, then S cannot output $\sigma = g^{x_3}$ because x_3 has been selected independently from x_1 and x_2 (except with negligible probability).

This is in violation of the assumption that DDH is hard in $\mathbb{G}$. Therefore, $\delta(\kappa)$ must be negligible. □