Trust-Based and Privacy-Preserving Fine-Grained Data Retrieval Scheme For MSNs

Enahoro Oriero*, Khaled Rabieh*, Mohamed Mahmoud†, Muhammad Ismail ‡, Erchin Serpedin‡, and Khalid Qaraqe‡

*Department of Electrical & Computer Engineering, Tennessee Tech University, Cookeville, TN, USA
†Electrical and Computer Engineering, Texas A&M University at Qatar, Doha, Qatar
‡Electrical and Computer Engineering, Texas A&M University, College Station, USA

Abstract—In this paper, we propose a trust-based and privacy-preserving fine-grained data retrieval scheme for mobile social networks (MSNs). The scheme enables users to create a log of trusted users who store (or are interested in) some topics related to a subject of interest. A subject is a broad term that can cover many fine-grained topics. In creating logs, we leverage friends-of-friends relationships and transferrable trust concept. Each user trusts its friends and the friends of friends. If a friend is not interested in a subject, he can help his friend in creating the log by linking the friend to his friends without knowing the subject to preserve privacy. In order to reduce the storage and computation overhead, we use Bloom filters to store the topics. A distinctive feature in our scheme is that it can query users who possess a fine-grained topic, rather than querying users who are interested in the broad subject but they may not have the specific topic of interest. We analyze the security and privacy of our scheme and evaluate the communication and computation overhead.

Index Terms—Trust; privacy preservation; Mobile Social Networks; and Data Retrieval.

I. INTRODUCTION

With the rapid increase in the number of mobile devices, the applications of Mobile Social Networks (MSNs) are becoming very popular. MSNs allow users to discover and interact with existing and potential friends [1]. They involve real-time communications to enable information sharing and social interaction amongst users. Many promising applications of MSNs require exchanging information on a subject of interest. Users who are interested in a subject do not necessarily possess the same topics of interest. That is why it is not sufficient to know the users’ subjects but also their topics of interest. In this paper, the term “subject” refers to the broad category of interest, like “Soccer”, “Politics”, etc., while the term “topic” refers to a sub-category under the subject. Usually topics are many and fine-grained, e.g., the subject “Soccer” can have topics like “World Cup 1998”, “Spanish League 2015”, etc.

Our scheme target three applications called chatting, file sharing, and Web page pre-fetching. In chatting applications, users need to contact other users about a topic of interest. For instance, if a user wants to ask someone who has experience in repairing cars, it should first look for someone who is knowledgeable in the subject “Car Repair”. Then, it may need to know fine-grained information about users’ experience like “Air Conditions Repair” or “Changing Tires”. These are fine-grained topics under the subject “Car Repair”. In file sharing applications, each user stores a group of files, e.g., for songs, pictures, etc., and users can request them. In such applications, if the subject is “Songs”, topics can be the names of the stored songs. In Web page pre-fetching applications, when a mobile device is connected to Wi-Fi, it downloads the Web pages that will be most likely requested by the user, so that the downloaded pages will be used when the mobile device does not have Internet connectivity. However, it is possible that a user needs a Web page that has not been downloaded. In this case, one can contact nearby friends to request the Web page. For instance, in this application, if the subject is “Politics”, a topic is the requested Web page. However, security and privacy issues in these applications are becoming a real concern.

In this paper, we propose a trust-based and privacy-preserving data retrieval scheme For MSNs. The scheme addresses security and privacy issues such as how to securely identify a trusted user who is interested in a subject, how to prevent others who are not interested in a requested subject from knowing the subject, and how to securely communicate and exchange information with a friend who possesses a topic on the subject of interest. First, we discuss a trust-based and privacy-preserving log creation scheme to enable users to build a log of trusted users who store some topics related to a subject of interest. In order to preserve privacy, if a user is not interested in a subject, he/she cannot know the requested subject by the scheme. Also, distrusted users cannot know the subject even if they are interested in the subject. In creating the log, we leverage friends-of-friends relationships and transferrable trust concept. Each user trusts its friends and the recommended friends by his/her friends. When a user looks for users who are interested in a subject, it does not only search in its list of friends but also the friends of friends. If a friend is not interested in the subject, he/she can help his/her friend by linking the friend to his/her friends without knowing the requested subject. The scenario can be extended to include more friends of friends. The motivation here is that collecting topics only from direct friends may not be enough and users may need a wide range of topics.

Our scheme enables the users to share symmetric keys with friends of friends. It uses Bloom filters to reduce the storage and computation overhead of the users’ topics. This is important because the number of topics may be large. After creating the log, it can be used to request a topic from the users who possess it. The data retrieval in our scheme is fine-grained because users exchange fine-grained topics, rather than querying users who have the subject of interest, but they may not have the topic of interest. We analyze the security and
privacy of our scheme and evaluate the communication and computation overhead.

The remainder of this paper is organized as follows. Section II presents the considered network and threat models. Section III discusses the proposed scheme. The security and privacy analysis and performance evaluation are presented in Section IV. The related works are discussed in section V, followed by the conclusions in Section VI.

II. NETWORK AND THREAT MODELS

Our network model is made up of users and an offline-trusted authority (TA). Each user has a group of trusted friends. It should maintain a trust value for each friend based on their interactions and if a friend’s trust value degrades to a threshold, the friend is no longer considered trustworthy and should be removed from the list of friends. The TA computes and distributes keys and secrets to users. We assume that each user can communicate to his/her friends. The friends can be physically close to the user and ad hoc networking technology can be used to communicate. Also, friends may be far from the user and they can communicate via the Internet.

For the threat model, we assume that friends are honest but curious. They do not aim to disrupt the proper operation of the scheme, but they are only curious to collect some private information such as the requested subject or topic. Friends do not share the secrets distributed by the TA because this may lead to exposing their private information such as subjects and topics of interest. However, outside adversaries may try to infer private information and disrupt the scheme, e.g., by launching attacks like packet replay, impersonation, etc.

III. PROPOSED SCHEME

In this section, we first present the bootstrap phase in which a trusted authority distributes cryptographic credentials to the users. This phase should be run only once when the application is first launched. Then, we explain our privacy-preserving and trusted-based log creation scheme. A log is created for each subject and it should have trusted potential sources for data retrieval and their topics of interest. Finally, in privacy-preserving data retrieval phase, the log is used to contact the trusted sources to request fine-grained topic.

A. Bootstrap

An off-line trusted authority bootstraps the system as follows. It selects a large prime number \( q \) and creates a finite field \( Z_q^* \) of order \( q \). \( G_1 \) is an additive group with generator \( P \) and order \( q \). \( G_2 \) is a cyclic multiplicative group with the same order \( q \). \( \hat{e} \) is a bilinear pairing function that maps elements in \( G_1 \) to elements in \( G_2 \), where \( \forall \ P_1 \) and \( P_2 \in G_1 \), \( \hat{e}(P_1, P_2) \to G_2 \).

The bilinear pairing function has the following properties:

1) Bilinear: \( \hat{e}(aP_1, bP_2) = \hat{e}(P_1, P_2)^{ab} \) for all \( P_1, P_2 \in G_1 \) and \( a, b \in Z_q^* \).
2) Non-degenerate: There exists \( P_1, P_2 \in G_1 \) such that \( \hat{e}(P_1, P_2) \neq 1 \). In other words, the map does not send all pairs in \( G_1 \times G_1 \) to the identity in \( G_2 \).

3) Computable: There is an efficient algorithm to compute \( \hat{e}(P_1, P_2) \) for all \( P_1, P_2 \in G_1 \).

Let \( h() \) and \( H() \) be two one-way hash functions, where \( h(): \{0, 1\}^* \to Z_q^* \) and \( H(): \{0, 1\}^* \to G_1 \). The trusted authority chooses a random element \( sk_i \in Z_q^* \) and computes \( PK_i = \frac{1}{sk_i}P \), where \( sk_i \) and \( PK_i \) are its private and public keys, respectively. For each user \( i \), the trusted authority chooses a random element \( sk_i \in Z_q^* \) and computes \( PK_i = \frac{1}{sk_i}P \), where \( sk_i \) and \( PK_i \) are the user’s private and public keys, respectively. The trusted authority generates a certificate for each user to bind its public key to its identity. A typical certificate should have the user’s identity, public key, expiration date, and the trusted authority’s signature. Using the certificates and signatures, each user can prove that he/she is a legitimate member. The trusted authority should issue, renew and also revoke the certificates if necessary. For every user to join the network, it should contact the trusted authority to receive the necessary credentials. Finally, the trusted authority publishes \( \{Z_q^*, G_1, G_2, q, P, PK_i, H(), h(), \hat{e}\} \) and keeps \( sk_i \) secret.

Each user has a group of subjects of interest. Examples of subjects are “Politics”, “Sports”, “Songs”, etc. Each subject has many underlying fine-grained topics. For instance, a subject “Politics” can have underlying topics like “US 2016 Presidential Election”, “US Civil War”, “World War II”, etc. In case of file sharing applications, if a user has the subject “Songs”, the topics are the names of the songs the user stores. In case of chatting applications, if the subject is “Car Repair”, topics can detail the user’s experience, e.g., “Air Conditioners Repair”, “Engine Repair”, etc. In case of Web page pre-fetching applications, if the subject is “Politics”, a topic is the Web page, e.g., http://www.cnn.com/2015/09/07/europe/europe-migrant-crisis/. The trusted authority distributes a secret \( s_i \in Z_q^* \) to each user who is interested in subject \( i \). The secret will be used to check whether a user is interested in the subject to protect privacy.

B. Privacy-Preserving and Trust-Based Log Creation

The objective of this phase is to enable a user to create a log for a subject of interest. The log stores the trusted users who have the subject and their fine-grained topics. When the user wants to retrieve some data, it uses the log to know whom it has to contact to request the data. In this subsection, we focus on building the log of a subject, and in the next subsection, we will discuss how the log can be used to retrieve data. In creating a log, we target several objectives. First, in order to preserve privacy, if a user does not have the subject, he/she should not be able to know the subject of the log that is being created. The second objective is to include only trusted users in the log. To do that we use the concept of transferable trust, i.e., not only the direct friends are trusted but also the friends of friends.

The notion of trust used in this paper is defined as the degree of belief or the probability that a user will act in a certain way in the future based on the past interactions. Trust values are calculated from the past interactions to predict the expected future behavior. For instance, If a user misbehaves a lot in the
past, there is a strong belief that this user will most probably misbehave in the future, and thus such users should be avoided. Trust is time-sensitive, so users should periodically evaluate their friends’ trustworthiness, i.e., a user’s trust value at time $t$ may be different from its value at another time $t'$. Each user should maintain a list of trusted friends. If a friend does not cooperate in sharing data or gives false data, he/she will be deleted from the list. Considering trust in data retrieval applications as a continuous value rather than binary [2]. Each user should periodically evaluate their friends' trustworthiness, i.e., a user’s trust value at time $t$ can be different from its value at another time $t'$, and thus such users should be avoided. Trust is time-sensitive, so users should periodically evaluate their friends’ trustworthiness, i.e., a user’s trust value at time $t$ may be different from its value at another time $t'$. Each user should maintain a list of trusted friends. If a friend does not cooperate in sharing data or gives false data, he/she will be deleted from the list. Considering trust in data retrieval applications as a continuous value rather than binary [2]. Each user should periodically evaluate their friends' trustworthiness, i.e., a user’s trust value at time $t$ can be different from its value at another time $t'$.

To create a log for a subject of interest, Fig. 1 shows that user $A$ should send Log Request ($LReq$) packet to his/her friends. If a friend (like $B$) has the requested subject, it replies with Log Reply ($LRep$) packet. If a friend is not interested in the subject (like $C$), he/she should not know the subject to preserve privacy, but it can help his/her friend by forwarding the request to his/her friends. If one of his/her friends has the subject (like $F$), it replies with $LRep$ packet. It can also establish a symmetric key with $A$. To restrict the transmission area of the $LReq$ packet, it has Time to Live (TTL) field. TTL is decremented at each node and once it reaches zero, the packet is not forwarded further. For instance, in the figure, TTL is two at $C$ and one at $D$, $E$ and $F$, so the packet is forwarded more.

The details of the exchanged packets to create a log are given in Fig. 2. User $A$ computes and sends a $LReq$ request to his/her friend $B$. The packet has $A_1$, $A_2$, TTL and its signature $\sigma_A$, where $\sigma_A = sk_A H(A_1, A_2, TTL)$, $A_1 = r_A K_{AB} P$, $A_2 = \hat{e}(P, P)^{r_A}$, $r_A$ is a random element in $Z^*_p$, $s_r$ is the secret of the requested subject, and $t_s$ is a timestamp. User $B$ checks the time stamp to ensure that the packet is fresh. Replayed packets should be dropped. Then, it verifies the signature by checking whether $\hat{e}(\sigma_A, PK_B) \equiv \hat{e}(H(A_1, A_2, TTL), P)$. The proof is as follows:

$$\hat{e}(\sigma_A, PK_B) = \hat{e}(sk_A H(A_1, A_2, TTL), PK_B) = \hat{e}(H(A_1, A_2, TTL), \frac{1}{sk_A} P) = \hat{e}(H(A_1, A_2, TTL), P)$$

Then, $B$ verifies whether it has the subject requested by $A$ or not as follows: $\hat{e}(A_1, h(s_r t_s)^2 P) \equiv A_2$. For the subject’s secret, $B$ should try all the subjects it has. The equation will only hold when $B$’s subject ($s_r$) equals to the subject requested by $A$ ($s_j$). If $B$ has the same subject requested by $A$, it sends back $LRep$ packet; otherwise, it tries to help $A$ by forwarding $LReq$ packets to its friends, as indicated in Fig. 2. The $LReq$ packet has $B_1$, $B_2$, the encryption of the log of interest stored at $B$ and its signature, where $B_1 = r_B K_{AB} h(s_r t_s)^2 P$ and $B_2 = \hat{e}(P, P)^{r_B}$. $A$ checks whether $B$ has the requested subject as follows: $\hat{e}(B_1, h(s_r t_s)^2 P) \equiv B_2$. $A_1$ and $A_2$ can prove to $B$ that $A$ is interested in the subject and $B_1$ and $B_2$ can prove to $A$ that $B$ is interested in the subject. $A$ uses the log received from $B$ and other users to create its log. Finally, after the log is created by $A$, it distributes the log to the users who sent their logs to enable them to update their logs.

As indicated in the figure, $B$ forwards the request to its friends if TTL is more than one. If $B$ does not have the requested subject, it should not know the subject. Only $B$’s friends who have the subject can know it. The packet has $A_1$ and $A_2$, where $A_1 = r_B K_{AC} A_1$ and $A_2 = A_2$. The user $C$ verifies whether it has the subject requested by $A$ as follows: $\hat{e}(A_1, h(s_j t_s)^2 P) \equiv A_2$. $C$ should try all the subjects ($s_j$) it has. The equation holds only if $s_j = s_r$. If $C$ has the same subject requested by $A$, it sends to $A$ $LRep$ packet that has $C_1$ and $C_2$, where $C_1 = r_C K_{AC} h(s_r t_s)^2 P$ and $C_2 = \hat{e}(P, P)^{r_C}$. $K_{AC}$ is a shared key between $A$ and $C$ and $C$ calculates it as follows: $\hat{e}(A_1, A_2)$.

Note that $A_1$ and $A_2$ can prove to $C$ that $A$ has the
subject and \( C_1 \) and \( C_2 \) can prove to \( A \) that \( C \) has the subject. Similarly, \( A \) calculates the shared key (\( K_{AC} \)) as follows: \( C_2^{\prime, r} \). Then, it verifies \( C_1 \) and \( C_2 \) as follows: \( \hat{e}(C_1, \frac{h(s, t, e_{AC}^2)}{K_{AC}}) = C_2 \). The equation holds if \( C \) has the subject requested by \( A \).

A user’s log should have the users who are interested in the subject, the shared key with each user, and a Bloom filter having all the topics each user has. Bloom filter is used to reduce the storage and communication overhead because it can store a large number of topics compactly. The number of topics is expected to be large because topics are fine-grained.

The Bloom filter is a data structure that can compactly store a set of elements in a bit vector. Before adding items to the vector, all the bits are set to zero. In order to add an element \( I_i \) to the filter, the element should be hashed using \( K \) different hash functions. The resultant hash values point at locations in the bit vector. All the locations pointed by the \( K \) hash values should be set. Moreover, to check if an element \( X \) is in the filter, it should be hashed with the \( K \) hash functions. If all the locations pointed by the hash values are set, \( X \) is in the filter with a certain probability; otherwise, \( X \) is certainly not in the filter. It is possible that an element is not in the filter but all the locations pointed by the hash values are set because they are set by other elements by accident. This case is called false positive. We refer to [3], [4] for more details about Bloom filter. The probability of the occurrence of a false positive is

\[
P_f = (1 - (1 - \frac{1}{m})^{KN})^K,
\]

where \( m \), \( K \), and \( N \) are the size of the vector, the number of hash functions, and the number of elements stored in the filter, respectively.

### C. Privacy-Preserving Data Retrieval

When user \( A \) wants to retrieve some data, it should check the relevant subject’s log to know whom it has to contact, e.g., user \( C \). Then, as indicated in Fig. 2, it sends a Data Request (DReq) to \( C \) that responds with Data Reply (DRep) packet. The DRep packet should have either the requested data or “File is not found” encrypted with the shared key. The file may not be found because of the false positives of the Bloom filters. This case is called a “failed request”. User \( A \) has to send a request to another user who has the topic (if there is). If the user sends the data, this is called a “successful request”.

The Bloom filters can be designed to reduce the probability of failed requests. A request failure probability is equal to the Bloom filter’s false positive \( (P_f) \), and a request’s success probability \( (P_s) \) is

\[
1 - P_f = 1 - (1 - (1 - \frac{1}{m})^{KN})^K
\]

As mentioned above, a user may need to contact several users until he/she can retrieve the data. In order to assess the performance of the data retrieval scheme, we use the probability of retrieving the data at most by \( n \) trials. In the rest of this subsection, we will explain how the Bloom filter can be designed to determine this metric. The probability of successfully retrieving the data at most by \( n \) trials is

\[
\sum_{i=1}^{n} P_f^{i-1} \times P_s,
\]

assuming that all the filters have the same \( P_s \) and \( P_f \).

Fig. 3 gives the probability of successfully retrieving data at most by \( n \) requests at different values of the Bloom filter size (\( m \)) in bits, where the number of topics stored in each filter is 100 and the number of hash functions (\( K \)) is five. It can be seen that the probability increases with the increase of \( n \). For example, when the Bloom filter’s size is 250 bits, the probabilities of successfully retrieving data at most by 3, 5 and 7 trials are 0.87, 0.96, and 0.99, respectively. The increase of the filter’s size increases the probability of retrieving data at the same \( n \). It can also be seen that if \( m \) is set to 350, it is almost certain that the data will be retrieved at most by the third trial. Storing 100 topics in 350 bit Bloom filter means that each topic is stored in 3.5 bits. This is much more efficient than using traditional way by storing the topics in ASCII code that assigns eight bits per letter. For instance, if the average topic size is 20 letters, each topic will be stored in 160 bits.

Fig. 4 gives the probability of retrieving data at trial number \( n \). This probability should equal to \( P_f^{n-1} \times P_s \). It can be seen that the increase of \( m \) increases the data retrieval probability at \( n = 1 \) and decreases the data retrieval probability at \( n > 1 \). For example, when \( m \) is 500 bits, the probability of successfully
retrieving data at the first trial is 0.9 and it is 0.1 at the second trial. This indicates that a user can retrieve data after only one or two trials while keeping the storage and communication overhead very low.

For simplicity, we assume that all the Bloom filters of a subject have the same size and store the same number topics. In reality, users may store different number of topics, and this can create a scenario where the filters have different successful data retrieval probability, which can change the values of the performance metrics. To maintain the same values for the metrics when users have different number of topics, the users should change the Bloom filter size to keep the same successful data retrieval probability. When the number of topics increases, the Bloom filter size should increase to keep the same successful data retrieval probability.

IV. EVALUATIONS

A. Security and Privacy Analysis

Subject privacy: Eavesdroppers and friends who are not interested in the subject cannot use $A_1$ and $A_2$ to know the subject of interest because they do not know the subject’s secret $(s_i)$. Distrusted users who are interested in the subject cannot use $A_1$ and $A_2$ to know the subject because they do not have $K_{AB}$. Only trusted friends who have the subject can know the requested subject in the log creation packets, and this is necessary to create the log. The friends who do not have the subject can relay the log creation requests to their friends, and only the friends who are interested in the subject can know the request’s subject.

Mutual proof: The user $A$ sends $A_1$ and $A_2$ to prove to $B$ that he/she is interested in the subject, and $B$ replies with $B_1$ and $B_2$ to prove to $A$ that he/she is also interested in the subject. If $B$ is not interested in the subject, it cannot compute valid $B_1$ and $B_2$ but it can compute valid $A_1$ and $A_2$ to enable its friends to verify whether they have the subject of interest.

Unlinkability of log creation packets: If a user sends requests at different occasions for the same subject (to update the log), the requests look completely different and attackers cannot know whether a request is for a new subject or old one. This is due to using one-time random number $r_a$ in each request that can make $A_1$ and $A_2$ of the different requests look different.

Subject guessing attack: $B$ checks whether it has the subject requested by $A$ as follows: $\hat{e}(A_1, \frac{h(s_r, t_i)}{K_{AB}}) = A_2$. If $B$ does not have the secret of the requested subject $(s_r)$, it may try to guess it by trying different values for $s_r$ until the equation holds. $s_r$ should be long enough to make guessing it using exhaustive search infeasible.

Topics privacy: In creating a log, users encrypt the Bloom filters to preserve the privacy of the topics. If a Bloom filter is not encrypted, attackers cannot extract the stored topics because hash functions are one way, but they can try different topics to check if they are in the filter. Encrypting the filters can provide higher privacy preservation by preventing this. Only the log creation packet sender is able to decrypt the filter and know with a degree of uncertainty if his/her friend has a certain topic without disclosing all the friend’s topics.

Trust-based log creation and data retrieval: If a distrusted user is interested in the subject of a log creation request, he/she cannot know the subject because he/she does not have $K_{AB}$, i.e., only trusted users can open the log creation requests and respond to them. We used the concept of transferable trust. Each user maintains a list of trusted friends and trusts its direct friends and their friends as well. A friend can be removed from the list of trusted friends when it misbehaves, e.g., does not cooperate in sending data or send false information.

Replay attacks: Replayed packets can be identified and dropped due to using timestamps in log creation requests.

Attacks against key agreement: In our scheme, we have used a key agreement procedure to enable a log creation packet sender to share a key with a friend of friend. The key is used to secure the data request and reply packets. In the key agreement procedure, the computation of the key is not controlled by only one user, but the two users who execute the procedure contribute to the key. This usually produces a more robust key than computing the key by only one user because it may select weak random values. The procedure can also thwart Man-in-the-middle attacks by signing the key contributions, such as $\hat{e}(P, P)^{r_B}$ and $\hat{e}(P, P)^{r_A}$. This deprives the attacker from sending $\hat{e}(P, P)^{r_B}$ on behalf of $A$ or $C$ because he/she cannot compute the signatures.

B. Performance Evaluation

Communication and Storage Overhead: The communication overhead is measured by the amount of data (in bytes) transmitted in each packet. $G_1$, $G_2$, and $Zq^*$ have order $q$ that is 32 bytes. The elements of $Zq^*$ has 32 bytes but the elements of $G_1$ and $G_2$ have 64 bytes using elliptic curve cryptography. $A_1$ and signatures are elements in $G_1$ and $A_2$ is an element in $G_2$. Log creation request packet requires around 192 bytes. The size of the symmetric key $K_{AB}$ is 64 bytes. The size of the Log reply packet is 192 bytes + 64 bytes × the number of blocks in the log, where each block is 64 bytes. The storage overhead is the storage space required to store topics. Assuming that the average number of characters in each topic is 20 and each character takes 8 bits, 15.63 K bytes are needed to store 100 topics. Using Bloom filter, 100 topics can be stored in only 350 bits (0.34 K byte).

Computation Overhead: The computation overhead is measured by the time required to perform an operation in milliseconds (ms). We measured the computation times of the multiplication, pairing, and exponentiation operations using MIRACL cryptographic library [5] running on 2.00 GHz Intel processor and 2 GB RAM. Our measurements indicate that the multiplication, exponentiation and pairing operations take 0.65, 2.4, and 7 ms, respectively. Using these measurements, the computations of $A_1$ and $A_2$ need 1.3 ms and 9.4 ms, respectively. Similarly, the computations of $B_1$ and $B_2$ need 1.3 ms and 9.4 ms, respectively. The computation of the shared key needs 2.4 ms. The Total computation time of the LReq and LRep packets take 10.7 ms each. AES symmetric
key encryption/decryption takes 0.5 ms/block and signature verification operation takes 7 ms.

V. RELATED WORK

Although several privacy preserving schemes have been developed for different networks, such as vehicular ad hoc networks (VANETs) \[6\], \[7\], ad hoc wireless networks \[8\], sensor networks \[9\], \[10\], smart grid \[11]–\[14\], and cellular networks \[15\], \[16\], the privacy problem we address is different. In the literature, two different research areas are related to this paper, named, privacy preserving profile matching and content retrieval. Privacy-preserving profile matching schemes allow friend discovery and matchmaking while preserving the privacy of personal profile information. In privacy-preserving content retrieval schemes, files are usually stored in a server and the objective is to enable a certain set of users to retrieve the files \[17\]. The server cannot know the content of the files but it can apply an access policy. Our scheme differs from these schemes as we apply data retrieval in MSNs, where the data is stored in friends’ devices rather than a central server.

Numerous privacy-preserving profile matching schemes have been proposed for MSNs. Dong et al. \[1\] propose protocols to compute social proximity between two users to discover potential friends in a privacy-preserving manner. Rabieh et al. \[18\] proposed a privacy-preserving chatting scheme for VANETs. The scheme can be used in sharing common-interest information among drivers. Yi et al \[19\] propose a privacy-preserving information retrieval scheme using fully homomorphic encryption. Guo et al. \[20\] proposed a scheme in which users with identical interests establish social relationships amongst themselves for content dissemination.

VI. CONCLUSION

In this paper, we have proposed a scheme to enable users to build a log of trusted users that stores some topics related to a subject of interest. In creating the log, we have leveraged friends-of-friends relationship and transferrable trust concept. In order to reduce the storage and computation overhead, we used Bloom filters to store the users’ topics. It has been shown that performance metrics can be determined by properly designing the filters. Our evaluations and analysis have demonstrated that our scheme can achieve our security/privacy objectives, the use of Bloom filters can make the scheme scalable, and the computation and communication overhead is acceptable.

ACKNOWLEDGEMENT

This publication was made possible by NPRP grant # NPRP 6-415-3-111 from the Qatar National Research Fund (A member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

REFERENCES

\[5\] Miracl, “Multiprecision integer and rational arithmetic c/c++ library.”