WiFi Authentication through Social Networks
– a Decentralized and Context-Aware Approach –

Yunus Durmus Koen Langendoen
Delft University of Technology, The Netherlands
Email: {y.durmus,k.g.langendoen}@tudelft.nl

***Abstract*—With the proliferation of WiFi-enabled devices, people expect to be able to use them everywhere, e.g., at work, while commuting, and when visiting friends. In the latter case, home owners are confronted with the hassle of controlling the access to their WiFi router, and usually resort to simply sharing the password. Although convenient, this solution breaches basic security principles, and puts the burden on the friends who have to enter the password in each and every of their devices. The use of social networks, specifying the trust relations between people and devices, provides for a more secure *and* more friendly authentication mechanism.**

In this paper, we progress the state-of-the-art by abandoning the centralized solutions to embed social networks in WiFi authentication; we introduce EAP-SocTLS, a *decentralized* approach for authentication and authorization of WiFi access points and other devices, exploiting the embedded trust relations. In particular, we address the (quadratic) search complexity when indirect trust relations, like the device of a friend's kid, are involved. We show that the simple heuristic of limiting the search to friends and devices in physical proximity makes for a scalable solution. Our prototype implementation, which is based on WebID and EAP-TLS, uses WiFi probe requests to determine the pool of neighboring devices and was shown to reduce the search time from 1 minute for the naive policy down to 11 seconds in the case of granting access over an indirect friend.

***Keywords*—Social Devices, WiFi Authentication and Authorization, WebID, EAP-TLS, EAP-SocTLS**

I. Introduction

With the uprise of social networks like Facebook and Twitter, users have become dependent on the Internet with WiFi being the dominant access network technology due to its high capacity. This trend has put the burden of access control on home owners as visiting family and friends expect to be able to use the owner's WiFi Access Point (AP) as if at home, in the office, or while commuting. The quick fix is to share the AP's password with the visitors, who can then enter it in their WiFi-enabled devices (tablets, smartphones, etc.). This behavior compromises basic security, as passwords are shared with many, which is aggravated by the habit of selecting short passwords that are easy to crack. To reduce the burden, and associated risks, of manual access control it has been proposed to use the trust relations embedded in social networks to automate device authentication and authorization [1], [2]. A promising idea that deserves additional attention as the integration of the social network in the basic AP authentication mechanisms completely removes the need for password management. We will refer to such integrated systems as social WiFi APs.

Existing solutions for social network integration assign a centralized system to control the authentication process. For example, WIMAN [1], a young startup, lets users access WiFi networks by using Facebook Platform, a popular single-sign-on technology managed by a (large) pool of servers. A second example is Instabridge [2], who created its own centralized online social network, and used that to distribute the WiFi password among the friends of an AP owner. In a wider perspective, researchers are broadening the scope of online social networks by integrating even more devices. The Social Internet of Things [1] and social access controller [2] are two inspiring works that initiate the sharing of device functionalities like location information over centralized proxy servers. Through registration these centralized servers are made aware of the complete social network of the device owner and this information is used to safeguard the proper use of the integrated devices.

Although convenient and readily available, centralized approaches for creating social WiFi APs cannot be used offline (i.e., fail without Internet connectivity). Some of them are prone to single-point-of-failure, scalability problems and generally they raise privacy concerns. To address these drawbacks we advocate a decentralized approach in which individual APs take full control and perform the device authorization themselves; access will be granted when a trust relation can be established between the AP owner and the owner of the client device as recorded in a social network. Creating a *decentralized* social WiFi AP entails two main challenges:

- **system design:** the fundamental design questions are (i) how to integrate devices into (existing) distributed online social networks, and (ii) how to manage the associated credentials in a secure and decentralized way.
- **search complexity:** as information is scattered across devices searching for a (transitive) trust relation between AP (owner) and (client) device will incur communication delays. Crawling the social network needs to be optimized as the search space grows exponentially with the length of the trust chain.

The first challenge, system design, is addressed by leveraging the WebID standard [3]. WebID uses well-known ontologies like Friend-of-a-Friend (FOAF) that are designed to solve the interoperability problem among online social networks [4]. Instead of shared secrets (i.e. passwords), X509v3 certificates are used for authentication. These certificates are adapted to connect the devices they represent to a social network by including a link (URI) to a social profile on the web. Both devices and humans must publish their social relations (owners, friends) in these profiles to allow for an integrated

[1] www.wiman.me
[2] www.instabridge.com

solution. In Section III we will detail how we adapted the *Extensible Authentication Protocol-Transport Layer Security (EAP-TLS)* authentication framework to work with the WebID certificates and social profiles. The resulting system is dubbed EAP-Social TLS (EAP-SocTLS).

The second challenge, search complexity, is addressed by a context-aware approach to bound the search space to probable candidates when searching for indirect relations like the smartphone of a friend's child. Limiting the number of candidate friends linking the AP and the device is crucial as verifying credentials and collecting profiles is expensive. We employ the simple heuristic that it makes most sense to check only those devices (and owners) who are in the direct vicinity of the AP. After all, it is highly unusual for people to grant access to random acquaintances if not accompanied by a "known" mutual friend. This intuition is corroborated in [5], where it is shown that the likelihood of having a social connection to a person is inversely proportional to actual distance to that person. In our EAP-SocTLS implementation we track regular IEEE 802.11 probe requests to determine which devices are in the vicinity, and sort them according to time of arrival, checking the most recent ones first. This strategy was experimentally verified to greatly reduce the search time; for a social network of neighbor degree 4, the worst case performance of an indirect friend search was decreased from one minute to a mere 11 seconds.

In summary, the contributions of the work presented in this paper are as follows:

- we describe the first decentralized social WiFi AP design freeing the average home owner from the burden of manual access control (see Section III).
- we introduce the use of context information to reduce the search complexity for establishing indirect trust relations in a distributed online social network, and detail an implementation based on tracking standard probe request messages (see Section IV).
- we provide experimental results validating the feasibility of our design, and report on the effectiveness of using context information (see Section V).

II. Background Information

In this section, the WebID protocol, and WiFi probe requests are explained briefly. The WebID protocol is used for distributed social network integration and WiFi probe requests are used to detect the presence of direct friends in the vicinity.

Before explaining the WebID, we should explain a crucial semantic vocabulary called Friend-of-a-Friend (FOAF). FOAF is a vocabulary with which a person can publish its social network in a profile document on the web. An example is shown in Fig. 1. In this example, Alice declares her friendship to Bob and Charlie using "*foaf:knows*" statements. URIs are used to identify people instead of their plane names. It is a distributed alternative to the centralized social networks.

A. The WebID Protocol

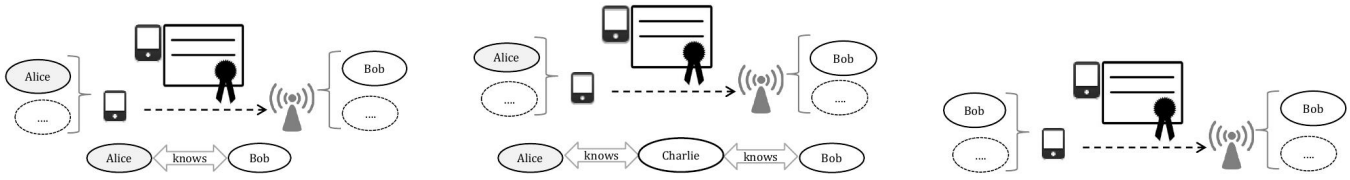
WebID is a *single-sign-on* and distributed authentication standard. It was designed for humans and distributed online social networks (DOSN). WebID uniquely identifies a person, company, organization or any other thing with a URI and the URI is placed in a X509v3 certificate. Its protocol for authentication and authorization works as embedded in

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf=
"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:pingback="http://purl.org/net/pingback/"
         xmlns:cert="http://www.w3.org/ns/auth/cert#">
  <foaf:Person rdf:about= "https://wonderland/alice/card#me">
    <foaf:name>alice </foaf:name>
    <foaf:givenName>alice</foaf:givenName>
    <foaf:img rdf:resource="https://wonderland/alice.png"/>
    <foaf:nick>alice</foaf:nick>
    <foaf:mbox rdf:resource="mailto:alice@wonderland.com"/>
    <foaf:knows rdf:resource="https://wonderland/BOB/card#me"/>
    <foaf:knows rdf:resource="https://wonderland/CHARLIE/card#me"/>
    <cert:key>
      <cert:RSAPublicKey>
        <cert:modulus rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#hexBinary">
          c2e98ceadf831ab308735c8b30e54a76fe76a9d6...
        </cert:modulus>
        <cert:exponent rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#int">
          65537</cert:exponent>
      </cert:RSAPublicKey>
    </cert:key>
  </foaf:Person>
</rdf:RDF>
```

Fig. 1. Example public profile of Alice. The profile contains the public key information via "cert:key" and the social network via "foaf:knows" declarations.

Transport Layer Security (TLS). It simply replaces certificate authority (CA) based certificate verification step of TLS with a social network based one. Instead of a CA, social profiles and trusts between the people verify a certificate. It resembles Web of Trust (WoT), but social networks are used to establish trust instead of people signing the certificates of their trusted friends. As a result, WebID with its protocol, enables us to involve social networks in authentication.

In the WebID protocol there are no passwords. A client claims its identity by just using a certificate. Initially, TLS ensures that client holds the corresponding private key for the public key in the certificate. Then the WebID protocol takes over the control. There are two steps: identity verification and trust. In the identity verification step, which is the authentication step, certificate verification is done with social web profiles. Certificate contains the URI address of the personal profile document of the client in the optional Subject Alternative Name (SAN) field. When the authenticating server receives the certificate, it follows the URI link in the SAN field and fetches the profile document such as in the example shown in Fig. 1. SPARQL, the semantic web query language, is used to query the profiles. Server checks the equality of the public key provided in the certificate and the personal profile document. If the public keys are the same, it concludes that the client is the owner of both the certificate and the personal profile document. The URI serves as the identity of the authentication requester; the name, surname or email address fields are not taken into account. Although the identity of the client is verified as the URI, the server still cannot trust the information presented in the profile document. Therefore, an authorization (trust) step is required. In this authorization step, authenticating server crawls every profile on web to discover a social tie between the client and the server. The social network ties are declared by the "*foaf:knows*" statements. As presented in Fig. 1, Alice declares that she knows Bob by using the "*foaf:knows*" statement. The time consuming part is this trust



(a) Direct Trust (b) Indirect Trust (c) Same Owner

Fig. 2. Different types of trust. Oval shapes represent the owners of the devices.

and authorization step where a social tie from the AP owner to the client device owner is discovered. After the identity verification and trust steps, TLS takes the control back and proceeds further to encrypt the channel with a symmetric key.

B. WiFi Probe Requests

Wireless client devices (supplicants) detect the APs by scanning the WiFi channels. There are two modes of scanning: passive and active. Passive scanning sniffs every channel for AP beacons, whereas active scanning sends probe requests to APs to check their presence. Active scanning is preferred over passive since it decreases the AP detection time, hence the duration for AP association. In the 802.11 standard the frequency and burst size of the probe requests is left to the vendor. The standard states that on every WiFi channel the supplicant sends one or more probe request and waits for the replies from APs for some time before moving to the next channel. In Section III-B, the analysis of inter-arrival times of probe requests from some devices show that with less than 200 *sec* APs get a probe request from a supplicant.

III. Decentralized Social WiFi Access Point

In this section we explain our proposal for decentralized social WiFi access point that uses distributed social networks for authentication. The WebID protocol replaces the certificate authority based certificate verification in TLS with a distributed social networks based one for HTTPS connections. EAP-TLS (RFC-5216) is a WiFi authentication standard that also uses TLS. We combine these EAP-TLS and WebID for our decentralized social WiFi AP and renamed EAP-TLS as EAP-Social TLS (EAP-SocTLS). With the EAP-TLS standard, both parties can authenticate the peer, however due to the replication of steps at both sides, we only explain the AP side. In EAP-TLS, the client device (supplicant) sends a certificate to the AP (authenticator) and the AP passes the certificate to the authenticating server which can be the AP itself (as in our case). The authenticating server verifies the certificate by checking the signature of the certificate authority. Then based on authorization rules the supplicant is allowed to join to the WLAN. The difference of EAP-SocTLS is the certificate verification and trust. The certificate is verified and the supplicant is authorized by the use of WebID and DOSNs. Certificates can be self-signed since only WebID verification is required. Authorization, which is based on trust to the supplicant, is established by the use of social networks.

Assume that the supplicant is a smartphone owned by Alice and it tries to access the AP of Bob. When the AP gets the certificate of the smartphone, first it verifies the certificate by following the profile URI in the certificate and checks if the keys "match". After verification, the AP learns the owner of the smartphone, Alice, by checking the social web profile of the smartphone. As the AP knows its owner, Bob, a trust (friendship) relation can be discovered now. There are three possibilities:

- **Direct Friends:** Alice and Bob are direct friends. In their social web profiles, they declare that they know each other by "*foaf:knows*" statements (Fig. 2(a)),
- **Indirect Friends:** Alice and Bob are not direct friends, but they have a common friend called Charlie (Fig. 2(b)). The AP should search the social profiles of Bob and Alice for Charlie or any other mutual friend,
- **Same Owner:** Bob owns both the smartphone and the AP and he tries to access to his own AP (Fig. 2(c)).

For the sake of simplicity, we assume that having a social connection to a person is enough for trust. However, in real life, we may not want to grant access to our complete social network to access our WLAN. The chain of access control rules should be involved in the process. The role of the supplicant's owner may become crucial in authorization. We left access control as future work. However, the AP still needs to crawl the DOSNs to find out relations that comply with these access control rules. Therefore, the social network search explained in the following section is still a valid problem.

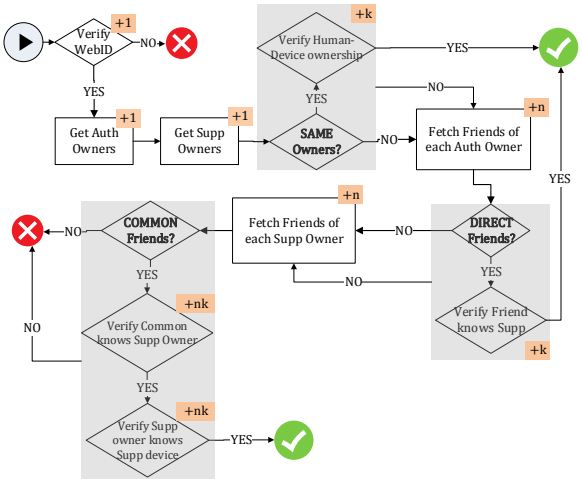


Fig. 3. Steps for searching social trust. In shaded areas, the set of friends/owners are checked serially in a loop which stops on the first success.

A. Social Network Search and Its Analysis

In the authentication and authorization process the exchange of the EAP messages are effected by the channel quality; but still, all the messages take so little time compared to *HTTPS* connections to the social networks (See Section V). The most time consuming and also varying part of the process is the trust, searching for relations on the DOSNs. Therefore, in this section we present an analysis of the search, and concentrate on it in the experiments.

A diagram of the search steps is given in Fig. 3. After certificate verification, first the *same owner* relation is checked. Since devices may have more than one owner, *same owners* can be multiple too. For each common owner, the possession of the supplicant is checked (possesion is declared by "*foaf:knows*" statements). In the first verified possession, the search completes successfully. Possession check is required to verify the device owner. For instance, in case of a stolen device, if possession is not checked, the "new" owner can access all the APs that the previous owner can. When there is no *same owner* relationship, the friends of each AP owner are fetched and compared to the supplicant owners to find *direct friend(s)*. In case of matches, all the direct friends are again checked for possession of the supplicant. In the absence of a trusted *direct friend*, the *indirect friend* relationship is checked. All the friends of the supplicant owner(s) are fetched and compared to the friend list of authenticator owners'. Then each common friend (direct friend) is verified if it knows the indirect friend.

Each *friend fetch*, *certificate verification* and *possession check* are separate *HTTPS* connections. To decrease HTTPS connections, the AP can store in cache the URIs of the owner(s) of the AP, and the URIs of the other devices with the public keys. Additionally, if there is enough storage, URIs of the direct friends can also be cached. However, it is not feasible to cache all the indirect friends due to their abundance (See Section V). If we assume that the number of owners is n and common friends/owners are k, then the worst case performances are detailed in Fig. 3 and summarized in Table I:

TABLE I. THE NUMBER OF HTTPS CONNECTIONS REQUIRED FOR AUTHENTICATION AND AUTHORIZATION WHERE n IS THE NUMBER OF THE OWNERS PER CLIENT AND SERVER, k IS THE INTERSECTION OF FRIENDS OR OWNERS DEPENDING ON THE CASE (SEE FIG. 3).

	Number of HTTPS connections	
	Without Cache	With Cache
Same Owner	$3+k$	---
Direct Friend	$3+n+k$	$2+k$
Indirect Friend	$3+2n+2nk$	$2+n+2nk$

Assume that a tablet has 4 owners (a small family), each of them has around 100 friends. In such a case $3+2\times4+2\times4\times k$, if k becomes 100 too, in worst case 811 HTTPS connections are required in the indirect friend case. If the device-human relation were many-to-one, devices might not have their own social profiles and the certificates of their owner might be placed directly into the device. Consequently, in the indirect friend case, the required number of HTTPS connections would have been $3+1\times k$ connections. However, to cover many-to-many relations between the devices and humans, the devices should have separate profiles at where their owners are defined. As a consequence, the search problem alters from linear $O(k)$ to quadratic $O(nk)$.

B. Collecting Presence Information

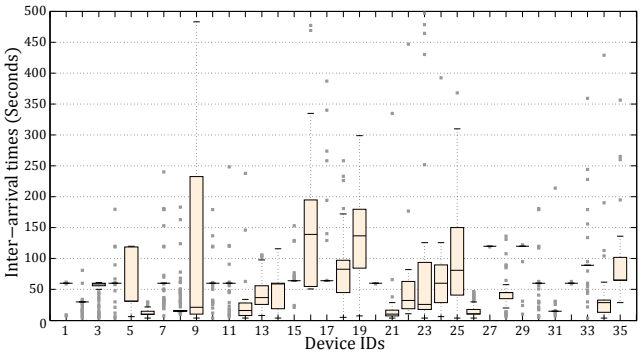


Fig. 4. Inter-arrival times of the probe requests from 35 devices in 2.4 GHz frequency band in two hours. Originally 87 devices have been recognized. However to have enough data to present statistical results, only the devices with more than 20 probe request readings are displayed.

As explained in the previous section, social network crawling for trust may take quadratic time. In Section V it is shown that even for a small social network, indirect friend search can take more than one minute. Intuitively, in WiFi authentication, direct friends who bridge the indirect friends to the AP, are probably in the close proximity at the time of association. We can bound the number of common direct friends by sensing the ones who are now in the vicinity of the AP. Moreover, by including the time of arrival, we can sort the common direct friends according to their temporal distance to the indirect friend.

We use WiFi probe requests (see Section II-B) to sense the direct friends in the vicinity of the AP. From previous authentications the AP stores the URIS of the direct friends with their devices' MAC addresses in its local cache. When the AP catches a probe request for the first time from a device or the device was absent for a while, time of arrival of the device is updated. When an indirect friend initiates the authentication, the time of arrival and the presence of direct friends are used to improve search performance. The crucial part in this context-aware system is the detection time of the devices in the vicinity. It is no use, if detection takes hours. As explained in Section II-B, there is no standard for frequency and burst size of probe requests. In [6], it is told that expected frequency is around 50-60 seconds. To verify their assumption we collected WiFi probe requests in an office environment for two hours. As shown in Fig. 4, due to the firmware of the WiFi card and also due to the channel noise, while some devices has almost constant intervals like the first device, some has extremely variable inter-arrival times like device $\#9$. The results do not indicate a common frequency range as in [6]. Nevertheless, we can still conclude that frequencies are less than 10 minutes and mostly even less than 200 seconds.

IV. IMPLEMENTATION

To validate our design of decentralized social WiFi APs and to determine the significance of sniffing probe requests to bound the search for indirect friends, we implemented a prototype version and tested it on real hardware. We altered the EAP-TLS source, part of Hostapd [3] as the basis for our EAP-

[3] http://w1.fi/hostapd/

SocTLS code [4]. In certificate verification part of the Hostapd code, we compare the public key to the one in social web profile. After verification, we call an external Python library [5] for the authorization part (i.e., searching for trust relations). Although all the extensions can be placed in Hostapd with C language, Python was selected for its ease-of-use. The lines of code required to complete EAP-SocTLS are about 450 for the Python library and 400 for Hostapd. An Sqlite3 database stores the probe requests. Note that our modification is completely transparent to the client side, who follows the normal EAP-TLS process when requesting service from the AP.

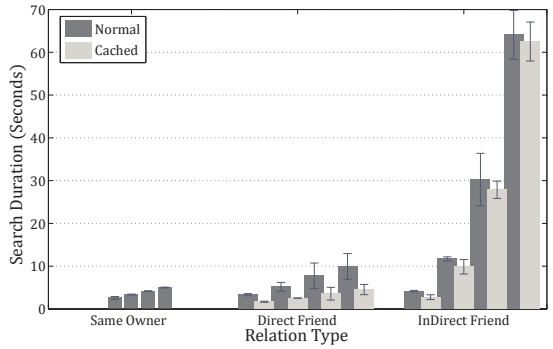
Our experimental setup consists of a Samsung Galaxy S2 smartphone (supplicant) connecting to a Dell Ultrabook with the Intel Centrino Advanced-N 6230 network card (802.11g) serving as the AP. For the tests, we created an artificial social network with uniform structure in the https://rww.io website, which is designed for semantic web applications and supports WebID. Although that web site has SPARQL support, for convenience our EAP-SocTLS implementation fetches complete profiles and processes them locally at the AP.

V. Evaluation

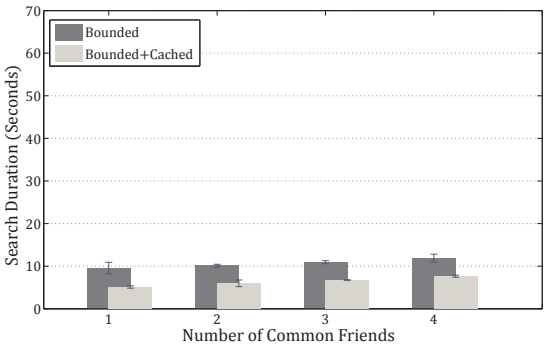
In the following experiments we show the performance of EAP-SocTLS, in particular with respect to the size of the social network (scalability). We study the three different trust types (same owner, direct/indirect friend), and vary the neighbor degree (number of friends/owners) from 1 to 4. In all scenarios we report worst case performance where all possible links (relations) are crawled. The number of common owners increases from 1 to 4 in the **same owner** case. In the **direct friend** case the number of owners on both sides increase, while each of the AP owners know all supplicant owners and vice-versa. In the **indirect friend** case the device owners and common friends increase together. Note that while the number of owners is increased in sequence for all types of trust, the effective number of common friends in the indirect friend case increases quadratically (from 1 to 16).

The duration required for legacy EAP messaging depends on the quality of wireless channel (inducing packet loss and retransmits) and was found to typically take less than one second. With respect to the duration of search for trust relations (Fig. 5) as implemented in the external Python library, EAP messages do not contribute much to the results. This prompted us to focus on the search part.

The duration of the search with different trust types and with increasing neighbor (friend) degree is given in Fig. 5(a), Following the analysis in Section III-A, the *normal* case indicates a linear increase in number of (same) owners and direct friends, and a quadratic trend for the indirect friend relation. We also present results for the case of an AP with a cache large enough to store the details of AP's owners and their respective friend lists. Usage of cache for the *same owner* case reduces the search time to zero as a simple lookup suffices; we do not plot these results for clarity. For the other trust types, the performance decreases a little, but the does not change the fundamental linear and quadratic behavior. One might argue that friends-of-friends information can also be cached, which



(a) Social relation search with increasing neighbor degree from 1 to 4.



(b) Indirect friend search with context Information.

Fig. 5. Durations for social relation discovery with different relations. Neighbor degree increases from 1 to 4 in Fig. 5(a) for each relation type. Neighbor degree is 4 in Fig. 5(b) while the number of common friends increase.

would bring down the search time to near zero, but the amount of storage requirement would then become prohibitive. In [7], for example, it is shown that in Facebook, friends-of-friends grow even faster than quadratic. For a person with 100 friends in Facebook, the number of unique friends-of-friends averages to 27500.

Observe that, even with caching enabled, the search time for indirect friends grows to over one minute for a neighbor degree of 4. In order to bound the indirect friend search, the AP collects the presence information as explained in Section III-B. To show the effect, we placed one direct friend in the vicinity and then add more direct friends linearly up-to 4. Fig. 5(b) shows that the use of context information has reduced the complexity for an indirect friend to a linear search.

Creating a large social network is costly. Therefore, we resorted to extrapolating the linear/quadratic search times to study the effects of scalability. In Table II, the estimated time required for searches with a neighbor degree of 10 and 100 are given. The naive Indirect Friend search becomes infeasible; even with degree of just 10, already 9 minutes are required. However, when applying context information only 12 seconds are needed. In real life we expect to obtains even better performances as we only reported worst case results. Moreover, search performance can be further improved by using more intelligent caching and parallel processing.

[4] https://github.com/yunus/Hostapd-with-WebID
[5] https://github.com/yunus/python-webid

TABLE II. PREDICTIONS FOR SEARCH DURATION OF HIGHER NEIGHBOR DEGREES. INDIRECT FRIEND RELATION IS ASSUMED TO BE QUADRATIC AND THE REST AS LINEAR.

	Worst Case Search Duration (Seconds)	
Neighbor Degree	10	100
Same Owner	9	82
Same Owner Cached	~ 0	~ 0
Direct Friend	23	224
Direct Friend Cached	10	96
Indirect Friend	537	5279
*Indirect Friend Bounded & Cached	12	88

*: Based on neighbor degree: 4; only number of common friends changes.

VI. DISCUSSION

With the proper use of caches, offline operation, without Internet connection, becomes possible at least for the same owner and direct friend relationships. For the indirect friend relationship, there must be an Internet connection.

Our decentralized social WiFi AP approach can be applied to any other authentication protocol that incorporates certificates such as WiFi Direct, SSH, IPsec. For instance, we have also created a WebID library [6] for AllJoyn peer-2-peer framework [7], which abstracts the communication layer for mobile devices. By including other systems, we can create a social network of devices where passwords are not used and all the authentication is automated.

In our current implementation, all the friendship information is assumed to be public for everyone, which may raise privacy concerns. To find the intersection of friend lists *private set intersection* techniques like *multi party computation* can be used [8]. However, to employ such techniques, end points at the web profiles should implement the SPARQL protocol. Then the endpoints are able to apply the techniques instead of just serving raw data.

VII. RELATED WORK

As briefly discussed in the introduction, WIMAN enables user access to WLAN by the Facebook Platform single-sign-on technique. It is a heavily centralized system, which based on only one type of social network, suffers from the single point of failure. Another company, Instabridge combines several social networks in their own centralized servers and distributes WLAN passwords to the friends of the device owner. It has an ambitious strategy to combine all the social networks and still suffers from a single point of failure. Differently from WIMAN, Instabridge supports offline operation however, password changes are costly. To revoke a password all the friends should be notified which entails quite a bit of overhead.

Like EAP-SocTLS there are other research efforts on socializing devices, which try to create social machine-to-machine access control. SenseShare [9] is one of the initial works, which is a common data store for sensor readings. All the sensors push their data to SenseShare, which stores and shares them with friends. SBone [10] and Social Access Controller [2] architectures transform the idea of sharing sensor readings to resource sharing. Both architectures have a central manager for access control. Our work, on the contrary, is the first social WiFi AP that incorporates a decentralized authentication procedure. With respect to privacy, centralized architectures can record all the accesses to the AP. Therefore, decentralization is required to protect privacy.

[6]https://github.com/yunus/WebIDforAllJoyn

[7]http://alljoyn.org

VIII. CONCLUSIONS

People share the passwords for their WiFi APs with their real-life social network. Smart devices can automate this process, and let friends access to WLAN without manual involvement. Centralized solutions have to deal with scalability, single point of failure problems and raise concerns on privacy. Our proposal of decentralized social WiFi APs uses the WebID authentication standard to connect devices to distributed social networks. Each device has its own social profile on the web, where it presents its owners. A significant challenge is the social network search complexity, which is quadratic for indirect friend relationships. To decrease the complexity, the set of direct friends who bridge the AP to the indirect friends are bounded using context information. That is, only friends in the direct vicinity are considered as derived from monitoring regular WiFi probe requests. The search space is bounded and sorted based on the existence and time of arrival of the direct friends. For a social network of neighbor degree 4, this heuristic decreases the search time from one minute to 11 seconds. Our design on top of the WebID protocol can be applied to any certificate-based authentication to automate the access control. However, it is crucial to come up with context-aware solutions to decrease the DOSN search complexity for trust.

ACKNOWLEDGMENTS

This work is supported by the Trans-sector Research Academy for complex Networks and Services (TRANS) project. We would like to thank Zekeriya Erkin and Alessandro Bozzon for their invaluable comments.

REFERENCES

[1] L. Atzori, A. Iera, G. Morabito, and M. Nitti, "The social internet of things (siot), when social networks meet the internet of things: Concept, architecture and network characterization," *Computer Networks*, vol. 56, no. 16, pp. 3594 – 3608, 2012.

[2] D. Guinard, M. Fischer, and V. Trifa, "Sharing using social networks in a composable web of things," in *PERCOM Workshops*, 29 2010-april 2 2010, pp. 702 –707.

[3] M. Sporny, T. Inkster, H. Story, B. Harbulot, and R. Bachmann-Gmur, *WebID 1.0, Web Identification and Discovery*, W3C Std., Rev. Editor's Draft, Dec 2011. [Online]. Available: http://www.w3.org/2005/Incubator/webid/spec/

[4] C. man Au Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-lee, "Decentralization: The future of online social networking," in *In W3C Workshop on the Future of Social Networking Position Papers*, 2009.

[5] L. Adamic and E. Adar, "How to search a social network," *Social Networks*, vol. 27, no. 3, pp. 187 – 203, 2005.

[6] M. Cunche, M.-A. Kaafar, and R. Boreli, "I know who you will meet this evening! linking wireless devices using wi-fi probe requests," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, 2012, pp. 1–9.

[7] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, "The anatomy of the facebook social graph," *CoRR*, vol. abs/1111.4503, 2011.

[8] E. Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear complexity," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science. Springer, 2010, vol. 6052, pp. 143–159.

[9] T. Schmid and M. B. Srivastava, "Exploiting social networks for sensor data sharing with senseshare," Posters, Center for Embedded Network Sensing, UC Los Angeles, Oct 2007.

[10] P. Shankar, B. Nath, L. Iftode, V. Ananthanarayanan, and L. Han, "Sbone: Personal device sharing using social networks," Technical Report, Tech. Rep., 2010.