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, be it at work, while commuting, or when visiting friends. In the latter case, home owners are confronted with the burden 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 solution 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 smartphone 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 and zero cost. 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 [4], [7]. 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, CISCO [1] and many other companies use captive portal [9] to let the users access to the network via social network login. A second example is Instabridge [2], who created its own centralized online social network, and uses 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 more devices. The Social Internet of Things [4] and social access controller [7] 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 and 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 manner.
- **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, we address by leveraging the WebID standard [13]. WebID uses well-known ontologies like Friend-of-a-Friend (FOAF) that are designed to solve the interoperability problem among online social networks [8]. 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] https://meraki.cisco.com/
[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, we address by a context-aware approach that bounds 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; setting up HTTPS connections for doing so takes time. 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 [3], where it is shown that the likelihood of having a social connection to a person is inversely proportional to the 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 1 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 detailing WebID, we must explain a crucial semantic vocabulary called Friend-of-a-Friend (FOAF), which can be used to establish distributed online social networks (DOSN) as an alternative to the centralized social networks. FOAF is a vocabulary with which a person can publish his 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. Note that URIs are used to clearly identify people instead of their plane names.

A. The WebID Protocol

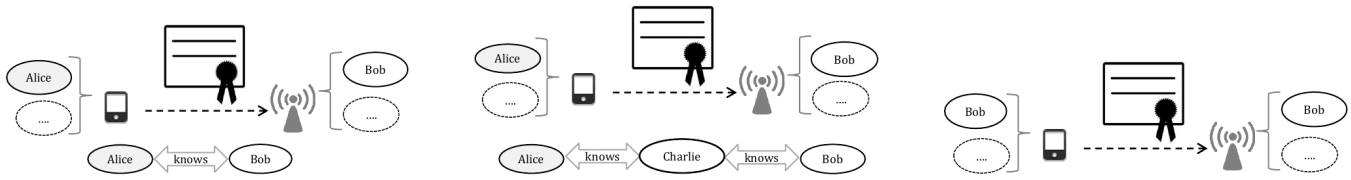
WebID is a *single-sign-on* and distributed authentication standard. It was designed for humans and DOSNs. WebID uniquely identifies a person, company, organization or any other thing with a URI and the URI is placed in a X509v3

```
<?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.

certificate. With the WebID certificate and a public social profile in FOAF vocabulary, a user can login to a server with WebID protocol. The protocol for authentication and authorization is designed for the Transport Layer Security (TLS) protocol. WebID simply replaces the certificate authority (CA) based verification step of TLS with a social network based one. Instead of a CA, social profiles and trust between the people verify a certificate. It resembles the 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 enables us to involve social networks in authentication.

When a client requests authentication by handing a certificate to the server, the TLS protocol starts off by verifying that the client indeed holds the private key that corresponds to the public key in the certificate. Then the WebID protocol takes over control for performing two essential steps: identity verification and trust establishment. In the identity verification step, which is the authentication step, certificate verification is done with social web profiles. A 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 the example shown in Fig. 1. SPARQL, the semantic web query language [10], is used to query the profiles. The 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. In case of a disclosure of a private key (e.g., through theft), the certificate can be revoked by simply removing its public key from the profile. 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 grant access without a social tie between the



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

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

client and the server. An authorization (trust) step is required. In this authorization step, the authenticating server crawls the profiles on the 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 of the WebID protocol is this trust and authorization step where a social tie from the server to the client is discovered. After the identity verification and trust steps, TLS takes control back and proceeds further to encrypt the channel with a symmetric key.

B. WiFi Probe Requests

Detecting which devices are present in the vicinity of an AP is key to bounding the search for indirect trust relations and WiFi probe requests indicate the presence of a wireless device. There is no need for developing custom solutions as we can simply leverage the normal WiFi registration process. 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 we will show that most devices probe the channels at least once every 200 sec providing an accurate view on the presence of the individual devices (and their owners).

III. Decentralized Social WiFi Access Points

In this section we explain our proposal for decentralized social WiFi access points that uses distributed social networks for authentication. We show that crawling distributed social profiles takes time especially for indirect relationships. Then we propose a context-aware heuristic to improve search performance that is based on presence information of the clients.

EAP-TLS (RFC-5216) is a WiFi authentication standard that also uses TLS. We combine EAP-TLS and WebID for our decentralized social WiFi APs and name it 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 access the WLAN. Our EAP-SocTLS differs in the certificate verification and trust. The WebID protocol is used for the certificate verification and the authorization of the supplicant. Authorization of the supplicant is established by trust relations embedded in social networks.

A device like an AP or a tablet can have multiple owners. If we would place the certificate of a person directly into a device, only one owner can be served. Therefore, we create a new <certificate, public profile> pair for the device. In the public profile, the device can present all its owners. An added benefit is that with individual profiles for devices we can create device-based access control rules. For simplicity, however, we only handle one generic access rule: if a profile mentions a friend, that friend can we be trusted.

Now we will show how authorization based on profiles works in our socal WiFi access point by means of an example. 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 embedded profile URI and checking 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 should 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 WLAN access to our complete social network. Therefore a chain of access control rules should be involved in the process. The role of the supplicant's owner may become crucial in authorization. We leave access control as future work.

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 establishing *HTTPS* connections to the social networks (See Section V). The most time consuming and also varying part of the process is searching for trust relations on the DOSNs.

Algorithm 1: Certificate verification and social tie searching. **k** and **n** are the number of owners and common friends, respectively.

```
   /* All the owners and friends are expressed as URIs */
   /* suppUri: URI of the supplicant */
   /* authUri: URI of the authenticator (AP) */
1  if not VerifyCertificate( suppUri) then                    // HTTPS
2      return FAIL
3  suppOwners = FetchOwnersOf( suppUri)                       // HTTPS
4  authOwners = FetchOwnersOf( authUri)                       // HTTPS
   /* SAME OWNER RELATION with complexity: k */
5  foreach so of Intersect(suppOwners, authOwners) do         // +k
6      if so isOwnerOf( suppUri) then                         // HTTPS
7          return SUCCESS
8      else
9          remove so from suppOwners
   /* DIRECT FRIEND RELATION with complexity: 2k */
10 authFriends = []
11 foreach ao of authOwners do                                // +k
12     friends = FetchFriends( ao)                            // HTTPS
13     foreach df of Intersect( friends, suppOwners) do       // +k
14         if df isOwnerOf( suppUri) then                     // HTTPS
15             return SUCCESS
16         else
17             remove df from suppOwners
18     push friends into authFriends
   /* INDIRECT FRIEND RELATION with complexity: k+2nk */
19 foreach so of suppOwners do                                // +k
20     friends = FetchFriends( so)                            // HTTPS
21     foreach idf of Intersect( authFriends, friends) do     // +n
22         if idf isFriendWith( so) then                      // HTTPS
23             if so isOwnerOf( suppUri) then                 // HTTPS
24                 return SUCCESS
   /* No connection was found. */
25 return FAIL
```

Therefore, we present an analysis of the search in this section, and concentrate on it in the experiments (Section V).

The algorithm of the search steps is given in Algorithm 1. Implicit *HTTPS* calls of the functions are highlighted by the comments. **k** and **n** are the number of owners and common friends, respectively. After certificate verification, first the *same owner* relation is checked. Since a supplicant or an authenticator may have more than one owner (e.g. a family AP), *same owners* can be multiple too. For each same owner, the possession of the supplicant is checked in line 6 (possession is declared by "*foaf:knows*" statements). At the first verified possession of the supplicant, the search completes successfully. The possession check is required to verify the supplicant owner. For instance, in case of a stolen supplicant, if possession is not checked, the "new" owner can access all the APs that the previous owner can. When there is no successful *same owner* relationship, the friends of each AP (authenticator) owner are fetched and compared to the supplicant owners to find a *direct friend*. In case of a match, the direct friend is checked for possession of the supplicant. From the nested loops in Algorithm 1, the direct friend relation seems to have $O(k^2)$ complexity. However, the complexity is actually "$2k \rightarrow O(k)$", since each direct friend is a supplicant owner and we do only 1 possession check per owner by removing it from the supplicant owners list (see line 17). 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 supplicant owner. The complexity of indirect search is "$k + 2nk \rightarrow O(nk)$".

TABLE I. THE NUMBER OF HTTPS CONNECTIONS REQUIRED FOR AUTHENTICATION AND AUTHORIZATION WHERE k IS THE NUMBER OF OWNERS PER SUPPLICANT AND AUTHENTICATOR (AP), n IS THE NUMBER OF COMMON FRIENDS (SEE ALGORITHM 1).

	Number of HTTPS connections	
	Without Cache	With Cache
Same Owner	$3 + k$	0
Direct Friend	$3 + 2k$	$2 + k$
Indirect Friend	$3 + k + k + 2nk$	$2 + k + 2nk$

Each one of the *friend fetches*, *certificate verification* and *possession checks* involves making a separate *HTTPS* connection. To decrease the number of HTTPS connections, the AP can cache the URIs of its owner(s), and the URIs of the friends of the owner(s) with their public keys. The friends list of an owner also involves the devices he possesses since both the friendship and the ownership are declared with the same "foaf:knows" statements. (This lack of discrimination has some detrimental effects, see Section VI, but can be ignored for now.) It is, however, unfeasible to cache all the indirect friends due to their abundance (see Section V). The worst case performances are detailed in Algorithm 1 and summarized in Table I. In worst case scenarios all the owners and friends have to be queried for possession and friend relations.

An indirect friendship search has quadratic $O(nk)$ complexity and as a consequence it can easily lead to huge numbers of HTTPS calls. Assume that a tablet and an AP both have $k = 4$ owners (a small family) and each of them has around 200 friends. If the number of *common* friends $n = 100$, according to Table I, $3 + k + k + 2 \times k \times n = 811$ HTTPS connections are required in the worst case. In Section V, we show that making such a high number of HTTPS connections requires to tens of minutes. To lower the latency, we exploit the presence information of the supplicants in the vicinity.

B. Collecting Presence Information

As explained above, social network crawling for trust may take quadratic time. In Section V it is shown that even for a small social network, an 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 close proximity at the time of association. We can bound the number of common direct friends by sensing the ones who are currently 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. The code for bounding and sorting can be added to line 21 of Algorithm 1.

We use WiFi probe requests (see Section II-B) to determine 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, the time of arrival of the device is updated. When an indirect friend initiates the authentication, the time of arrival and the presence of direct

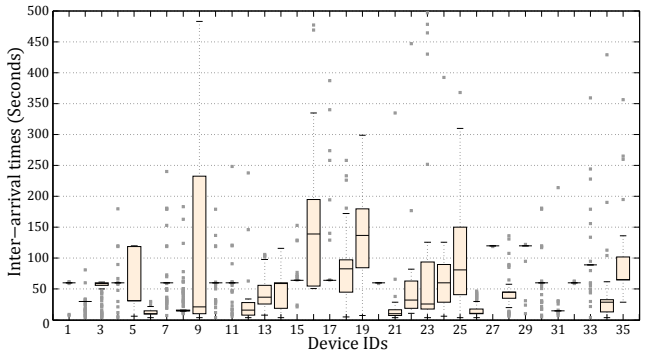


Fig. 3. Inter-arrival times of the 802.11 probe requests from 35 devices in 2.4 GHz frequency band over two hours.

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 of no use, if detection takes hours. As explained in Section II-B, there is no standard for interval and burst size of probe requests. In [6], it is stated that an expected interval is around 50-60 seconds. To verify that assumption we collected WiFi probe requests in an office environment for two hours. Fig. 3 shows the inter-arrival time distribution from 35 different devices. Overall 87 devices had been recognized. However to have enough data to present statistical results, only the devices with more than 20 probe request readings are displayed. The firmware of the WiFi card and also the channel noise determine the inter-arrival times. While some devices have almost constant intervals like the first device, some have extremely variable intervals like device #9. The results do not indicate a common inter-arrival range as in [6]. Nevertheless, we can still conclude that inter-arrival times are less than 10 minutes and mostly even less than 200 seconds. With the presence detection in minutes, proximity based common-friend sorting and bounding can work effectively.

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-SocTLS code [4]. In the certificate verification part of the Hostapd code, we compare the public key to the one in the 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 is C code, 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. The only requirement for the client device is to have a certificate with a link to the corresponding social-profile page.

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

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

[5]https://github.com/yunus/python-webid

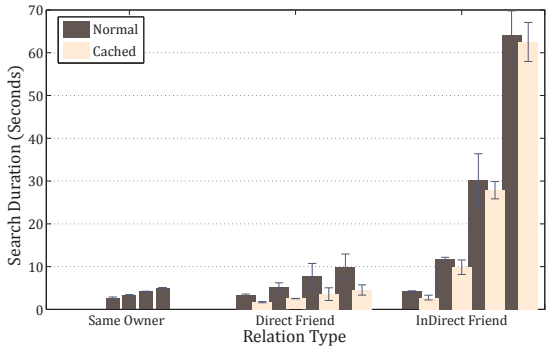
The integration of the WebID protocol into Hostapd was non-trivial due to Hostapd being large and complicated project in terms of lines of code and features. We embedded the WebID process in the OpenSSL *tls_verify_cb* callback function, which is invoked when an error is detected such as (in our case) an unfamiliar certificate failing the built-in CA-based verification. Our heuristic for speeding up the search process for indirect friends relies on presence information that is collected in the background (separate thread); All sniffed probe requests with the MAC addresses and timestamps are pushed to an SQLite3 database. This information is used to filter the common-friends by their temporal distance to the current supplicant being authorized. One challenge in the filtering is that OpenSSL does not pass the MAC address of the supplicant to the *tls_verify_cb* call. Fortunately, we are still able to access the MAC addresses of all the devices that are currently being authorized by Hostapd. Therefore, we use a pool of MAC addresses for the filtering in a round-robin fashion. Once we are done verifying a (in)direct trust relationship we give control back to Hostapd by returning from the *tls_verify_cb* function with the authorization result, and the protocol continues further with the original EAP-TLS sequence.

V. Evaluation

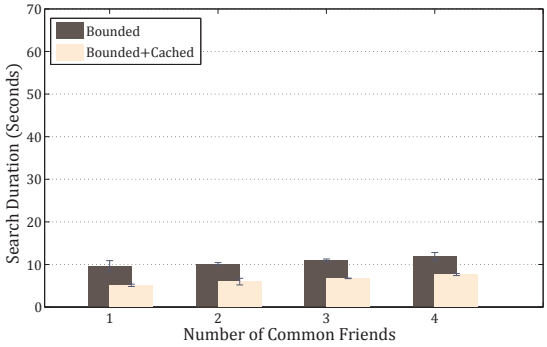
Our experimental setup consists of a Samsung Galaxy S2 smartphone (supplicant) connecting to a Dell Ultrabook with an Intel Centrino Advanced-N 6230 network card (802.11g) serving as the AP. For the tests, we created a social network with uniform structure at the https://rww.io website, which is designed 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.

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 with each of the AP owners knowing 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 for certificate and symmetric key exchange depends on the quality of a wireless channel (inducing packet loss and retransmits) and was found to typically take less than one second. With respect to the duration of searching for trust relations (Fig. 4) 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.



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



(b) Indirect friend search with context Information.

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

The duration of the search with different trust types and with increasing neighbor (friend) degree is given in Fig. 4(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 the 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 improves a little, but caching does not change the fundamental linear and quadratic behavior. One might argue that friends-of-friends information can also be cached, which would bring down the search time to near zero, but the amount of storage requirement would then become prohibitive. In [14], 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 27,500.

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 and exploits 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. 4(b) shows that the use of context information reduces the complexity for an indirect friend search to a linear process.

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 presence information only 12 seconds are needed. In real life we expect to obtain even better performance as we only reported worst case results. Moreover, search performance can be further improved by using more intelligent caching and parallel processing.

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. However, we should also note that caches may be stale and thus can lead to security issues.

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 the 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 authentication is automated.

In our current implementation, all 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 [5]. 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.

Lastly, in our proposal the profile document of a device lists its owners via *foaf:knows* links since there are no *foaf:owns* or *foaf:owned_by* relations in the FOAF standard or in any other vocabulary that we have encountered. However, the relation between the device and its user is not friendship, it is ownership. This lack of discrimination has a consequence in the effectiveness of the search for trust relations through (in)direct friends; our protocol also iterates over an owner's devices instead of just considering its trusted peers in the social network. As a future work, we plan to propose an ontology to

[6] https://github.com/yunus/WebIDforAllJoyn
[7] http://alljoyn.org

include the concept of ownership, and we consider refining that to the level of principal owner, second tier owner relationships to enhance search efficiency to the maximum possible.

VII. Related Work

As briefly discussed in the introduction, in captive portal [9], supplicant is allowed to join the WLAN, however kept in a walled garden until the user opens a browser and logins to a social network like Facebook. If the supplicant does not have a screen or browser software, it cannot access to the internet and consequently user cannot login. Moreover, captive portal is mainly a solution for public space hotspots where the WLAN service provider only wants to learn the identity of the user. There is no social network based access control that restricts users. A separate RADIUS server should be employed for access control. Another company, Instabridge combines several social networks in their own centralized servers and distributes WLAN passwords to the friends of a 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. SenseShare [11] 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 [12] and Social Access Controller [7] 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.

Apart from WebID, several other single sign-on (SSO) techniques are employed to eliminate the need for successive sign-ons. Among them OAuth [1] and OpenID [2] are the most popular. These techniques delegate the authentication to a trusted third party. Differently from them, WebID is a distributed protocol, and the trust depends on the social network avoiding the need for (shared) passwords. Besides, OpenID has recently been shown to be immature and include serious flaws [15] as a result from building it from scratch. Our approach instead builds on WebID, which itself is an enhancement over well-known and tested security protocols.

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 a person's 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 is bounded using context information. That is, we only consider friends present in the direct vicinity 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. The WebID protocol can be applied to any certificate-based authentication to automate the access control. However, like our probe request based restriction on indirect friends, it is crucial to design 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] Oauth 2.0. [Online]. Available: http://oauth.net/

[2] Openid. [Online]. Available: http://openid.net/

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

[4] 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.

[5] 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.

[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] 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.

[8] 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.

[9] B. Potter and B. Fleck, *802.11 Security*. O'Reilly Media, Inc., 2002.

[10] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. W3C. [Online]. Available: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[11] 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.

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

[13] 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/

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

[15] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services," in *Security and Privacy (SP), 2012 IEEE Symposium on*, may 2012, pp. 365 –379.