Protecting User Privacy in WiFi Sharing Networks

Till Elsner Denis Lütke-Wiesmann Björn Scheuermann
Heinrich Heine University Düsseldorf, Germany
{elsner, scheuermann}@cs.uni-duesseldorf.de, denis.luetke-wiesmann@uni-duesseldorf.de

***Abstract*—WiFi sharing communities are an interesting option for mobile Internet access. Today, however, users need to give up privacy in exchange for connectivity: they are required to expose their identity in order to be granted access. This information can be used to track users and to generate usage and movement profiles. We identify the challenges of providing a maximum level of privacy and anonymity in a WiFi community. From a discussion of privacy properties of existing architectures, we arrive at a privacy-aware WiFi sharing design for fully anonymous usage—without sacrificing any participant's security or legal safety. This system, like some previous designs, relays a mobile user's data via this user's home network; we discuss how and under which circumstances this contributes to privacy and anonymity. In order to enable users to remain fully anonymous in such a system, a means to locate the home network of a user without leaking permanent identifiers is needed. We introduce and evaluate *volatile host names*, a novel method to generate anonymized, non-persistent host names in standard DNS.**

I. Introduction

In a WiFi sharing community, users are allowed to access the Internet from foreign access points in exchange for sharing their unused Internet connection bandwidth with others. However, in today's deployed WiFi sharing communities (most prominently, FON [1]), this gain of connectivity comes at a significant cost in terms of privacy: in order to be granted access to the Internet via a community access point, a user is first authenticated by a central community operator. This provides the community operator with exact knowledge when and where a particular user accessed the Internet, yielding a detailed profile of her movement and community resource usage. Not only the community operator, but similarly also any community member providing an access point can learn about her users, and may even eavesdrop on the users' traffic.

Here, we consider the question how a WiFi sharing community can be built that provides full privacy protection for its users. Our key arguments, detailed in the course of this paper, are as follows. We start by discussing some WiFi community designs that have been proposed in the scientific literature, in particular [2], [3], [4], [5]. These proposals use triangle routing via the guest user's own home network. We will argue how this concept avoids the necessity to disclose the user's real-world identity to any other party, and therefore constitutes a promising basis for privacy-aware WiFi sharing.

However, even if the resulting design guidelines are followed and such a design is employed, significant risks for the users' privacy remain due to indirectly leaked identification attributes: we will discuss that if persistent attributes, even if completely unrelated to the real-world identity, make a user recognizable upon later returns, an eventual revelation of the user's real identity is very likely. This threat is not sufficiently taken into account by existing systems. In fact, in a WiFi sharing community based on triangle routing via the user's home network, it turns out upon closer examination that one central puzzle piece for providing full user privacy is missing: how can a user request to be connected to her home network without revealing an identifying attribute, namely the (dynamic) DNS name of this home network?

We are the first to solve this problem, with a mechanism which we call *volatile host names (VHNs)*. A volatile host name is a host name that can be resolved via standard DNS, but only once, i. e., for a very limited period of time. Multiple VHNs from the same user—referring to the same host or home network—cannot be linked. In combination with regularly changing IP addresses of the home network (which are common for many home Internet connections), a user repeatedly accessing the Internet via a community access point cannot be recognized based on this attribute. VHNs therefore provide the missing component that is needed to build a privacy-aware WiFi sharing community.

The remainder of this paper is structured as follows. In Sec. II, we examine previous work in the WiFi sharing context, with a focus on anonymity and privacy aspects. Based thereon, we identify the shortcomings of existing approaches, derive requirements for privacy, and discuss solution strategies in Sec. III. In Sec. IV, we describe our volatile host names approach, analyze its security, and evaluate its performance. We conclude the paper with a summary in Sec. V.

II. Related Work

The most popular deployed WiFi sharing community is FON [1]. Its centralized authentication scheme with user accounts bound to the user's real-world identity makes the FON community improper for anonymous use by design, as the community operator knows exactly which person used which access point at which point in time. From a user perspective, this is surely not desirable. As we will see, the problem can be overcome by avoiding centralized authentication.

Several designs in the scientific literature—including the proposals made by Sastry et al. [2], by Heer et al. [3], by Leroy et al. [4], and our own previous work [5]—route a mobile user's traffic via a trusted relay in her home network. For instance, this relay could be physically running on the user's own community access point. This concept is outlined in Fig. 1. The key motivation is to relieve the operator of the access point from the liability of potential misuse: since all traffic of a mobile user is routed via her home network,

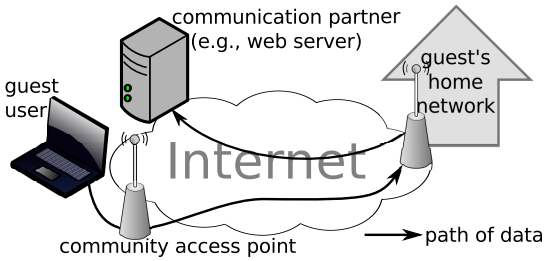

Fig. 1. WiFi sharing approach using triangle routing.

any potential misuse would also be attributed to this network, and not to the person offering the access point. Beyond this observation, however, we find that the "triangle routing" concept, if used in the right way, in addition also exhibits significant privacy benefits. We will therefore build upon this architecture when we discuss the construction of a privacy-aware WiFi sharing community in the following.

III. Privacy-Aware WiFi Sharing

In the remainder of this paper, we will step-by-step develop the requirements for a WiFi sharing system which allows its users to remain completely anonymous. At the same time, we will ensure that we do not sacrifice other important aspects like, e. g., the protection of access point providers against misuse of their service. To this end, we discuss privacy threats in WiFi sharing approaches, and how to overcome them.

A. Tracking by the community operator

A prototypical example for centralized authentication is the aforementioned community FON [1]. Binding a user's account to her real-world identity enables FON to identify a user in case malicious traffic arises. However, it also provides full knowledge about each user's movement and usage pattern.

As a means to avoid the central authentication instance, one might replace the online authentication process by certificate-based offline authentication. Instead of being directly authenticated by the community operator, guest users would then prove their community membership using a certificate issued by the operator. In the simplest case, this certificate would bind a public key for authentication to a real-world identity, and would certify that the certificate holder is a community member. The role of the community operator would be to issue these certificates. Since the community operator does not need to be contacted when the actual (public key-based) authentication is performed, the community operator can no longer track the usage patterns of community members.

B. Tracking by access point operators

The simple certificate-based scheme outlined above still reveals the identity of the users to the access point provider. One may very well argue that this is even worse than revealing the access pattern to a community operator, because the access point operator can not only observe when the access point has been used by which user: in addition, the access point can see the network traffic of each user.

Heer et al. [3] propose a system where the certificates do not include real-world identities. At a first glance, it may appear that, in such an approach, the access point operator does not know whom to hold accountable in case of misuse. This, however, is elegantly solved by a concept first proposed by Sastry et al. [2], where each user's traffic is relayed via the user's home network. A community access point does therefore not provide full, unrestricted Internet access to a user, but only access to her home network, which could be specified in the user's certificate. The home network forwards the data between the user's current location and the Internet. As can be seen in Fig. 1, the guest's communication partners on the Internet will thus see an address from her own home network as the source of the traffic. Consequently, there is no liability risk for the community member offering the access point. Moreover, such an approach can easily be combined with encrypting the traffic between the user's device and home network. This protects the mobile user's traffic from snooping by the access point, or by other users of the same access point.

C. Tracking anonymized identifiers

In a system as discussed so far, the community operator is not involved in the actual authentication, and the access point provider is not able to immediately link her users to real-world identities. While Heer et al. [3] term this property "anonymity", we will rather refer to it as "pseudonymity" as defined by Pfitzmann and Köhntopp [6]: while a connection can not directly be linked to the user's real-world identity, multiple connections of the same user can easily be linked to each other. Keeping real-world identities out of certificates does thus not suffice: the host access point gets to see the guest's certificate, which remains unchanged for each authentication. The host can thus easily determine if and when the same guest connected before.

Anonymity in the sense of *unlinkability* is important, because otherwise "anonymized" connection profiles about the holder of any certificate can be created, i. e., profiles which are not (yet) linked to a real-world identity, but only to a pseudonym/certificate. As these profiles include more and more data, the set of users that could possibly be the originator of the data becomes more and more constrained. In the context of anonymization services, this set of possible identities behind a pseudonym is termed the "anonymity set" [6]. Eventually—often surprisingly rapidly and easily [7]—the anonymity set becomes smaller and the real-world identity is revealed with very high probability. Once the real-world identity behind a given certificate is known, all past and future actions can be linked to this identity—that is, full tracking is possible.

D. WiFi sharing without persistent attributes

Avoiding this problem boils down to avoiding visibility of any persistent attributes of a user to the access point. This includes data exchanged during authentication, but also, for instance, the MAC address of the user's device: the same user must "look" entirely differently upon each connection.

As for the persistent information in the certificate, a solution strategy based on our own previous work [5] is possible. In this work, we introduced a WiFi sharing system without any central operator, not even a certification authority. The association between a user and this user's home network is not based on certificates, but is verified on-line by simply "asking" the home network. For details how this is accomplished we would like to refer the reader to [5]; the general idea is as follows: the guest user connects to an access point and provides it with the DNS name of her so-called *remote station*. The remote station is a specific host in the guest user's home network which is used to relay her traffic. The guest user shares a secret key with her remote station. The access point, before allowing any traffic between the user and the Internet, connects to the named remote station and performs a *remote station approval* handshake. The handshake provides the access point with an unencrypted and the guest user, via the access point, with an encrypted version of the same random number; if the guest user named the correct remote station, she will be able to decrypt the encrypted version and can thereby prove to the access point that data traffic between the guest and the named remote station can safely be allowed. The guest user will then be permitted to set up a tunnel to her remote station.

With respect to privacy aspects, the key feature of this handshake is that it does not reveal any information to the used community access point, except for the DNS name of the claimed remote station. In a typical setup, this will be a dynamic DNS host name [8] which resolves to the current IP address of the guest user's home network.

In summary, if we build a WiFi sharing community based on the triangle routing concept and adopt a remote station approval handshake along the lines of [5], only the following information about the guest will be accessible to the access point: (1) the DNS name of the user's remote station, (2) the current IP address of the remote station, as resolved via DNS, and (3) the MAC address of the guest user's device.

Recognition by MAC address can easily be avoided, as the assignment of a random MAC address prior to connecting to an access point is possible with every modern WiFi chipset and operating system. The issue of the IP address of the remote station, too, is easy to solve: a privacy-aware user should use an Internet provider where the IP address of her connection changes regularly, e. g., upon each re-connect. Such address changes could be triggered automatically by the provider on a daily basis (as is the case for a certain, but decreasing number of Internet providers), or, if not forced by the provider, by the user herself by shutting down and re-establishing her Internet connection, either manually or automatically. Since out-of-band channels like cellular connections may imply significant cost—especially when used abroad—, the most reasonable option to communicate the remote station's current IP address to the user is to announce it via DNS.

The key open issue that remains to be solved is therefore the following: upon connection establishment, the DNS host name of the remote station is handed over to the access point, in order to perform the remote station approval handshake; how can we avoid that this host name can be used as a permanent identifying attribute of the guest user? Note that the guest user cannot resolve the host name herself, because at this point she is not yet allowed to communicate with the Internet; moreover, allowing unauthenticated guest to perform name resolutions would also open up avenues for authentication circumvention techniques like DNS tunneling [9]. Note also that existing cryptographic DNS extensions like DNSSEC [10] are obviously not helpful here: DNSSEC provides authentication, but not confidentiality or anonymity.

IV. Volatile Host Names

Our solution to the problem of anonymizing DNS names is the *volatile host names* (VHN) approach. If a user decides to use volatile host names, she sets up a dynamic DNS account with a provider offering that feature. The user can then, in a first step, request an arbitrary number of so-called *request tokens* from her DNS provider (e. g., via a web site) prior to using the WiFi sharing community. Each request token can be used to construct a volatile (i. e., one-time) DNS host name for the user's home network. The user can download more request tokens at any time. Thereby, by registering a single dynamic DNS account, the user has an arbitrary number of DNS names for her home network available. The DNS provider does not need to store the tokens; that means, the VHN extension of the DNS server can provide an arbitrary number of DNS names to a user without the need to remember a single one of them.

In a typical usage scenario, a user contacts a community WiFi access point and would then need to name her remote station. At this point she would use one request token to generate a VHN. This VHN is then communicated to the access point as the remote station DNS name. The access point does not need to be aware that a VHN is used, as VHNs can be resolved using standard DNS, just like any other host name. By resolving the VHN, the access point will obtain the current IP address of the user's remote station. Soon later, the VHN will become invalid and can no longer be resolved. The user does not need to take care of this invalidation; she only needs to remove the corresponding requst token from the list.

To be practically usable, such a technique must, beyond anonymity, fulfill two important requirements. First, compatibility with standard DNS must be preserved—it is obviously not realistic to deploy updated DNS servers throughout the Internet. In our volatile host name system, the only server with extended functionality is the authoritative name server of the dynamic DNS provider, i. e., VHNs provide host name anonymization over standard DNS as specified in [11]. Second, the protocol should be constructed in a way that does not open up avenues for denial-of-service attacks against the DNS provider. This rules out a solution based on public key cryptography: the computational requirements would be far too large, and would make the DNS server susceptible to DoS attacks. VHNs use only symmetric cryptography so that, as we will show, the involved computations are very efficient. The abovementioned statelessness also contributes to making the approach robust against DoS attacks.

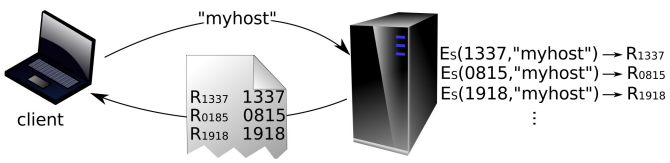

Fig. 2. Request token list generation.

A. Approach

Each user of a VHN-enabled DNS provider has a user ID c, which is basically a string and is only known to the provider and the client. In our prototype implementation, we used strings of either 24 bytes or 56 bytes (these sizes yield tokens of full cipher block size lengths). For each user ID, the DNS server stores the current IP address; this resembles the information stored by any standard dynamic DNS provider.

To be able to generate VHNs, a client must first obtain request tokens from the DNS provider, which the provider generates on demand using a symmetric key S. This key is *not* known by the user but *only* to the authoritative DNS server. S is also not user-specific: the same key S is used for all users (i. e., for all customers of the dynamic DNS provider). The generation of request tokens is outlined in Fig. 2. To generate a token, the server picks a random *transaction number (TAN)*; in our implementation, TANs are 8 byte long. It then encrypts the tuple (TAN, c) using key S. For encryption, the TAN and c would typically be concatenated. We denote the encrypted tuple by R. Both R and the unencrypted TAN are then transmitted to the client, together they form a request token.

To generate a VHN from a request token (TAN, R), the client first calculates a cryptographic hash value $cksum_t$ over both the current system time t and the TAN. We use an 160 bit SHA-1 hash for $cksum_t$; the timestamp is 4 byte long. The hash value serves as a checksum to prevent replay attacks. Then, the host name part of the VHN is assembled as the concatenation of R, t, and $cksum_t$. To be compliant with the range of allowed characters for DNS names as stated in [11], this string is then base32 encoded [12]. Finally, the domain name of the VHN-aware dynamic DNS provider is appended. In summary, the VHN looks as follows:

$$\mathrm{base32}(R\,t\,cksum_t).\texttt{example.com},$$

where `example.com` is the domain of the dynamic DNS provider. Base32 encoding adds 40 % in size, so for field lengths as stated above the length of the host name stays significantly below the limit of 255 characters in DNS [11].

Note that the domain name is appended in clear text to the scrambled host name, and the overall construction looks like any valid DNS host name. Thus, neither the access point nor any intermediate DNS servers need to be aware that a VHN is being used. In fact, they could not even tell a volatile host name apart from a "normal" (yet, admittedly, long and rather random) one. This trait also implies that no user would be forced to use a specific VHN provider (or to use VHNs at all)—each user is free to choose any service that is able to provide DNS names for the current remote station IP address.

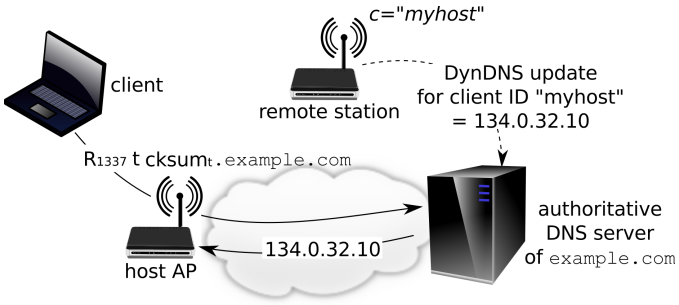

Fig. 3. Anonymized request.

Figure 3 sketches the way a VHN resolution request takes. When the authoritative DNS server of a VHN dynamic DNS domain receives a request to resolve a host name, it extracts R, t, and $cksum_t$. It decrypts R using S and thereby obtains c and the TAN. Using t and the TAN, the server can check whether $cksum_t$ is correct. If this check holds, the server compares t and its own current system time, to check if t is sufficiently current. Only then, the request will be answered. This requires synchronized clocks between a client device and the authoritative DNS server, but only to a very coarse accuracy that will easily be achievable in practice: a tolerance of a few minutes seems appropriate. The larger the allowed deviation, the longer will it be possible to resolve the VHN.

B. Security analysis

Without knowledge of the key S, it is not possible to extract the client ID c from a VHN. It is also not possible to decide whether two different VHNs belong to the same user.

An attacker could still keep track of the current IP address by repeatedly resolving a once obtained host name, enabling linking of different connections of the same user even if they use different host names. Such replay attacks are warded off by the inclusion of the timestamp t and its checksum $cksum_t$: t forces the invalidation of the VHN after a short time. Updating t would also require an update of $cksum_t$. The TAN required to do so cannot be obtained from the VHN by an attacker.

Also important to the security is the use of the cipher block chaining mode (CBC) for the encryption of R. To avoid unnecessary overhead, we use a fixed initialization vector (IV) for the encryption in all request tokens. This prohibits the use of an operation mode such as cipher feedback mode (CFB) [13]: the first ciphertext block with CFB encryption is simply an XOR of the encrypted IV and the first plaintext block. Any user of the same dynamic DNS server could thus easily obtain the encrypted IV from her own request tokens, because she knows the plaintext block. On this basis, she could recover the plaintext TAN from the VHNs of other clients. Using CBC for encryption avoids this problem.

C. Performance

The volatile host names approach has been designed with anonymity and security in mind, but, to avoid being prone to denial-of-service attacks, it also takes speed and protection of server resources into account. The server does not need to

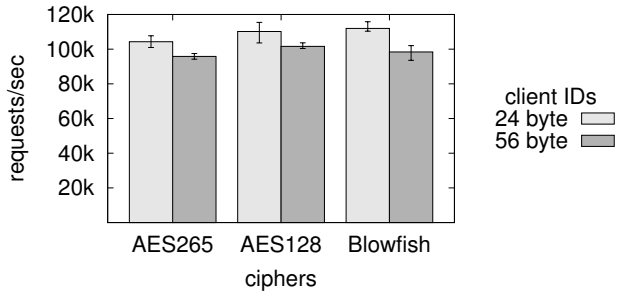

Fig. 4. Measured processing speeds.

store any data beyond the usual dynamic DNS information (i. e., the current IPs) and one single key S—no matter how many request tokens have been generated (or have been used). Generating a request token requires one symmetric encryption. Generating a VHN from a request token on the client side requires the calculation of one cryptographic hash value. Resolving a VHN takes one symmetric decryption plus the calculation of one cryptographic hash value. Using symmetric cryptography only, all these operations are highly efficient.

The only additional step required to resolve a VHN in contrast to a conventional DNS name is the *transformation* of the VHN to the encapsulated host name, i. e. the decryption of the VHN and the validation of the timestamp. Therefore, our evaluation focuses on this transformation, since all other factors influencing the resolution process performance also exist in conventional DNS and are not affected by the VHN scheme. To evaluate the transformation's performance, we have implemented VHNs in a prototype. It is written in Java, using the Java Cryptography Extension (JCE) with JCE Unlimited Strength Jurisdiction Policy Files 6. The evaluation has been performed using an 2.66 GHz Intel Core2 Quad CPU, with 4 GB RAM, on 64-bit Gentoo Linux (kernel 2.6.31).

We performed multiple measurement runs with different parameters. We used different symmetric block ciphers and client IDs c of length 24 byte and 56 byte. We took measurements using AES with key lengths of 128 and 256 bit, and Blowfish with a key length of 56 bit. Fig. 4 shows the number of requests that could be decoded within one second, for each of the abovementioned configurations. All results show the average over 10 runs and min-max error bars.

Using 24 byte long client IDs c results in R values with a length of 56 characters (24 byte client ID + 8 byte TAN, base32-encoded). The base32 encoded timestamp needs 7 characters. For the hash value we use a 160 bit SHA-1 hash, which grows to 32 characters through the base32 encoding. The complete VHN therefore has an overall length of 95 byte, not including the domain name of the dynamic DNS provider. During transport, at least 56 bytes of headers will typically be added (10 byte DNS + 8 byte UDP + 20 byte IP + 14 byte Ethernet frame + 4 byte Ethernet CRC), resulting in a lower bound on the overall size of a request packet of 151 byte.

One million requests, as used in our test, would thus result in at least 1.125 Gbit of traffic. Our prototype needs approx. 9.5 seconds for verifying and decrypting this number of requests, so it is capable of handling DNS traffic at line speed for connections of more than 120 Mbit/s. If 56 byte long client IDs are used, the results are even more favorable: despite a slightly lower number of decryptions per second (due to the longer ciphertext), the additional characters result in an even higher data volume (and thus a lower number of requests per second received on a given network link), so that this combination could operate at even higher line speeds.

In summary, our results show that even with our non-optimized Java implementation, processing VHN DNS requests is easily possible at typical line speeds. Therefore, thanks to being built on the basis of purely symmetric cryptography, our scheme is very efficient, and its low cryptographical overhead will not make the dynamic DNS provider's server susceptible to DoS attacks.

V. Conclusion

In this paper, we presented techniques applicable to provide full anonymity and privacy within a decentralized WiFi sharing community. We argued that the key aspect that needs to be solved is the avoidance of permanent identification attributes that could be used to recognize a guest. We showed that a combination of a triangle routing-based community network architecture, the certificate-free remote station approval handshake, and dynamically varying MAC and IP addresses is a good basis for such a system. In order to become fully robust against tracking by anonymized identifiers, it needs to be combined with a way to obtain the current IP address of a user's home network without using a permanently resolvable DNS name. We therefore developed volatile host names as a means to perform DNS request anonymization with low overhead and without a compatibility breach.

References

[1] "FON," Website. [Online]. Available: http://www.fon.com
[2] N. Sastry, J. Crowcroft, and K. Sollins, "Architecting citywide ubiquitous Wi-Fi access," in *HotNets '07*, Nov. 2007.
[3] T. Heer, S. Götz, E. Weingärtner, and K. Wehrle, "Secure Wi-Fi sharing on global scales," in *ICT '08*, Jun. 2008.
[4] D. Leroy, M. Manulis, and O. Bonaventure, "Enhanced wireless roaming security using three-party authentication and tunnels," in *U-NET '09*, Dec. 2009.
[5] W. Kiess, T. Elsner, B. Scheuermann, and M. Mauve, "Global grassroots WiFi sharing," in *WCNC '10*, Apr. 2010.
[6] A. Pfitzmann and M. Köhntopp, "Anonymity, unobservability, and pseudonymity – a proposal for terminology," in *Workshop on Design Issues in Anonymity and Unobservability*, Jul. 2000, pp. 1–9.
[7] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *SP '09*, May 2009.
[8] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound, "Dynamic updates in the domain name system (DNS UPDATE)," RFC 2136, Apr. 1997.
[9] T. van Leijenhorst, K.-W. Chin, and D. Lowe, "On the Viability and Performance of DNS Tunneling," in *ICITA '08*, 2008.
[10] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS security introduction and requirements," RFC 4033, Mar. 2005.
[11] P. V. Mockapetris, "Domain names – implementation and specification," RFC 1035, Nov. 1987.
[12] S. Josefsson, "The base16, base32, and base64 data encodings," RFC 4648, Oct. 2006.
[13] "Modes of operation for an *n*-bit block cipher," ISO/IEC 10116:2006.